Planar Separator Theorem

Aleksandar Timanov
RWTH Aachen University
aleksandar.timanov@rwth-aachen.de

***Abstract*—Divide and conquer is a widely spread strategy for solving complex problems efficiently. It is probably known to the reader from some of the base algorithmic approaches for sorting of lists like merge sort and quick sort, which both recursively divide the list into smaller ones, sort these ones, and then merge the results in order to get the final sorted solution. In this paper we are going to explore some of the laws by which a separator is defined, find efficient ways to calculate one and some of its applications.**

I. Introduction

In graph theory and more specifically when talking about planar graphs such an approach can be very useful in many situations. In the case of graphs the vertices of the graph have to be partitioned in such sets A, B and C that no vertices in A and B share edges in the graph. C would be the set of vertices, which divides the graph into two parts when removed, called the separator. The subgraphs formed after the separation have to be significantly smaller or in some cases with equally big costs, if the graph is weighted, than the whole graph. Therefore we have set the rule that no component of the graph after the separation should have cost over $2/3$ of the cost of the whole graph or in other words no component can be more than twice as bigger than the other one. This ensures that we have a suitable enough separation so that the subproblems can be solved using far less time and resources than the one we have started with. Because we remove the vertices in the set C from the graph when we divide it, often times an error has to be considered when using the approach to solve problems. So more often than not one can only approximate a solution of the problem. How good this approximation is relies on many factors like the number of the vertices, the distribution of the costs or other parameters which can be included in order to suit the algorithm better to the problem at hand.

There are many applications of the planar separator theorem that we are going to discuss in this paper. As already mentioned many complex problems can be approximated in far less runtime. Some algorithms for approximation of NP-complete problems, that apply the divide and conquer strategy recursively on the given graph, can terminate in polynomial times, which often times allows for several approximations to be found and compared to each other in order to calculate the error in the same time, the whole problem would have been solved without separating the graph. Other applications can take advantage of the strategy in a recursive manner in order to use the separator for something, because it is defined in such a way that its vertices always lay around the center of the graph. By finding the separators of the formed subgraphs recursively, a lot of problems can be solved in an efficient manner.

In the following section (II) we are going to begin by defining some ground rules and notation which will help us later on. After that in section III we are going to talk about some basic theorems and lemmas and finally define and prove the planar separator theorem (Theorem 4). In the IV section we are going to go through an algorithm for efficient calculation of a separator in a given planar graph. In section V we are going to show some applications of the theorem like solving NP-complete problems like the maximal independent set problem and embedding data structures. We are going to finish with a small conclusion in the end.

II. Definitions

Throughout the paper we are going to use the notation $G(V, E)$ for graphs, where V is the set of vertices and E is the set of edges connecting these vertices. The number of vertices in a graph would be referred to as n. An edge (v, w) is called incident to v and w and the vertices v and w are called adjacent to each other if such edge exists. A path with endpoints v and w ,which has length k, is a sequence of vertices $v = v_0, v_1, ...v_k = w$ such that (v_{i-q}, v_i) is an existing edge for every $1 \leq i \leq k$. Such path is called simple when one vertex comes more than once in the sequence and if the start and endpoint are the same vertex, then the path is also a cycle. The distance from one vertex to another one is defined as the shortest path between them.

A subgraph $G_1(V_1, E_1)$ of $G(V, E)$ is such a graph that $V_1 \subseteq V$ and $E_1 \subseteq E$. G_1 is a generalized subgraph of G, if again $V_1 \subseteq V$, but this time each of the edges of G_1 are being mapped with a function f into the set of paths in G, such that for every edge $(v, w) \in E_1$, the path $f(v, w)$ has endpoints v and w and no two paths, mapped with the function, share vertices except in the case when the shared vertex is an endpoint of both paths. A graph $G(V, E)$ is induced by a vertex set V_1 (where it has to apply $V_1 \subseteq V$) when we remove all the edges which are not connecting two vertices of V_1. Shrinking an edge means that we delete it as well as its endpoints and substitute them with a new single vertex which then has to be connected with the same vertices, the former endpoints of the shrunken vertex were connected. A degree of a graph is the maximal number of incident edges to a vertex in it.

Now we are going to explain the more important types of graphs. In a connected graph one can draw a path from each pair of vertices. A tree is a special type of graph which does not have any cycles. Every tree has a root vertex r. Every

vertex v, which lays on a simple path from the root to another vertex w, is called an ancestor of w (w is the descendant respectively), if the distance between v and w is just one edge then v is the parent of w and w is the child of v. The radius of the tree is the maximal distance a vertex can be from the root. A spanning tree in a graph is such a tree which covers every vertex of this graph.

A planar graph is such a graph that can be embedded (drawn) in the plane with no crossing edges. Each planar graph divides the plane in different sections which are called faces. A triangulation of a planar graph is the process in which edges are being added to the graph, if needed, until every face of the graph is a triangle.

III. Theorem

In this section we are going to take a look at some cases of graphs, whose separator can be found efficiently, and find out, if there is a general theorem which allows for every graph to be separated in a suitable way by gradually expanding the spectre of observed graphs and applying different techniques.

First we need to establish some ground rules in order for our further proofs to make better sense.

A. A closed curve divides the plane (Jordan curve theorem [1])

Theorem 1. *If a closed curve C on the plane is given, then removing all the elements that are building the curve C, would leave the rest of the components of the plane divided in two groups. The regions that can be distinguished are the "inside" and "outside" of C.*

B. Kuratowski's theorem of planarity [4]

Theorem 2. *Given any graph $G(V, E)$, we can find out, if the given graph is planar or not by examining, if it contains either a complete graph of $n = 5$ (K_5) or a complete bipartite graph ($K_{3,3}$), which has 2 sets with 3 vertices each, as a generalized subgraph. If neither of the two is found then we can conclude that the graph G is planar.*

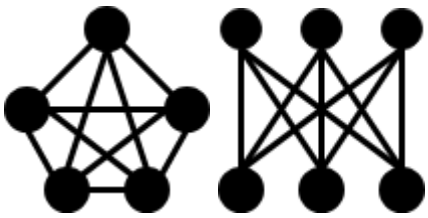

Figure 1. Here are shown the complete graph of 5 vertices K_5 (on the left) and the complete bipartite graph with two sets of 3 vertices $K_{3,3}$ (on the right)

C. Shrinking preserves planarity [5]

Theorem 3. *Given any graph $G(V, E)$, we can shrink any edge of G to a single vertex and the planarity will be preserved. We can prove the theorem by contradiction.*

Proof: Let us claim that the graph was planar before the shrinking of one of its edges to a vertex, but the graph G', which we get after that, is non-planar. Using Theorem 2, that we have talked about already, we can conclude that G contains neither K_5 nor $K_{3,3}$ and G' contains one of them. Because G' is a generalized subgraph of G, as it is constructed by shrinking one of G's edges, and G' contains one of Kuratowski's subgraphs, but G does not, we can conclude that our assumption is wrong, because we have been led to a contradiction.

By applying this logic inductively on many vertices, we can shrink whole connected subgraphs to a single vertex without making the new graphs non-planar.

D. Spanning trees as a tool for separation [5]

Lemma 1. *Given any planar graph G with non-negative vertex costs, which sum up to no more than one. G contains a spanning tree, whose radius is r.*

We can construct partitioning of the vertices of G into three sets A, B and C can be made. Here C is the so called separator, it should be as small as possible so that it is easier and faster for an algorithm to calculate it afterwards, and A and B are the two subgraphs (sub-problems), which have to be roughly equal to each other in the best case and significantly smaller than the initial problem. In order to satisfy this conditions, we strive to keep A's and B's costs no bigger than $2/3$ of the whole cost of G and we can prove that using a spanning tree, this can be achieved while the size of C would contain no more than $2r + 1$ vertices.

Proof: In case, there is a vertex with cost, exceeding $1/3$ of the total cost, then the lemma is true, because we can choose, for example, A to contain only this vertex, B to contain every vertex which does not have a direct edge to the one in A and C to be every other one. If there is a vertex, whose cost even exceeds $2/3$, then it has to be included in C in order to satisfy the lemma.

If no such vertices exist, then the first step of finding our partitioning is to embed the graph in the plane and triangulate it by adding edges. Then we can observe that every non tree edge forms a simple cycle with some of the tree edges. That is possible, because the spanning tree covers every vertex of the graph and no matter which two vertices, the chosen of us edge connects, we can follow the tree edges out of them to the root in order to construct the cycle. We can also see that such a cycle can not include more than $2r + 1$ edges (Twice the radius, if the endpoints of the chosen non-tree edge are on the boundary of the graph and we also have to count the root). Now we can claim that there exists at least one such cycle, whose removal would divide the embedded in the plane graph into suitable parts. Now we have to prove that this claim is also true.

We have to find such a non-tree edge (x, z), which minimizes the bigger cost between the inside or the outside of the simple cycle it forms. If two non-tree edge's cycles separate the vertices equally good, then we choose the one whose cycle

contains less faces. If we still have two cycles, which are equally suitable then it does no longer matter which one we will choose and we can choose arbitrarily. This would ensure that the two parts are as equal in cost as possible. Observing the different possibilities for such cycles we can conclude that neither cost of the sides of the separated graph according to them would be more than $2/3$, as that would always violate the rules, by which we have chosen the non-tree edge (x, z) and we would always have a better choice for it.

Why is this lemma important?: Now that we have seen that every graph, which contains a spanning tree can be separated in an efficient matter, it is only a matter of finding a spanning tree in order to calculate the separation of a graph.

E. Layering [5]

Lemma 2. *Given any connected graph G, whose vertices are evaluated to have costs, whose sum does not exceed one. The vertices are also partitioned into levels according to their distance from some vertex v, which is chosen beforehand. Here applies the following notation: $L(l_i)$ is the number of vertices on the level l_i, which is i-edges away from v, r would be the maximum distance from v and finally we are also going to add an additional layer at distance $r+1$ which does not contain any vertices, but is going to be used in our proof. We have to find the two layers l_1 and l_2, for which applies that the vertices in the levels from level 0 to l_1 and those on levels between l_2 and $r+1$ do not exceed in cost $2/3$.*

Once we have satisfied these rules we can conclude that a partition of G's vertices into the sets A, B and C exists that there are no edges connecting vertices from A and B, the cost of vertices in both A and B is at most $2/3$ and finally that C contains no more than $L(l_1)+l(l_2)+max\{0, 2(l_2-l_1-1)\}$ vertices.

Proof: In order to prove the lemma we need to take a look at all the possibilities for graphs it can be applied on.

The first option is that we have the two layers l_1 and l_2 overlapping each other, which means that $l_1 \geq l_2$. In this case the lemma is true, because we can simply assign A to contain all vertices between levels 0 to $l_1 - 1$, we know that their cost is under $2/3$ form the rules of the lemma, B would contain the levels beyond $l_1 + 1$, so from $l_1 + 1$ to r, and finally C would contain all vertices that were on the level l_1, whose count is equal to $L(l_1)$ which is obviously less than $L(l_1)+l(l_2)+max\{0, 2(l_2-l_1-1)\}$.

The second variant is that we have $l_1 < l_2$, in which case we have to make some adjustments to the graph in order to prove the lemma. Firstly we are deleting all the vertices on both layers l_1 and l_2, which leaves the graph divided into three parts. Because of the rules, by which l_1 and l_2 have been chosen we know that the outer most and the inner most part can not exceed $2/3$ in cost, but the middle part can have a cost which is more than $2/3$.

In the case, that the middle part is with lesser cost than $2/3$ of the whole cost, then we let A contain all vertices on this part of the three, which has biggest cost, B would have those that are on the remaining two parts and C would get the vertices on levels l_1 and l_2. The cost of the vertices in A would be under $2/3$, B's cost in the worst case (when the three parts all have the cost of $1/3$, would also be $2/3$, and C would contain $L(l_1) + L(l_2)$ vertices, which again is less than $L(l_1)+l(l_2)+max\{0, 2(l_2-l_1-1)\}$.

If the middle part is with bigger cost than $2/3$, we again have to make some changes to the graph. We delete all the vertices on level l_2 and above, we shrink those on levels l_1 and below to a single vertex, which preserves planarity (Theorem 3). Because the layers were assigned we know that there is a path to every vertex, which starts in v, that means that the newly formed graph also contains a spanning tree and it has a radius of $l_2 - 1 - l_1$. Now that we have found a spanning tree we can apply lemma 2 and we will get as a result the three sets A^*, B^* and C^*. Now we can assign the vertices in A^* or B^*, depending on which has the higher cost, to A, C would get those on levels l_1 and l_2 as well as the ones in C^*, and B would be assigned the rest of G's vertices. The worst case here would be that the middle part (from l_1+1 to l_2-1) takes the whole graph. In such case we have already seen in the proof of lemma 2 that neither of the sets A and B would have cost greater than $2/3$, C would contain the vertices on C^* which are equal to twice the radius plus one, but because after the shrinking of the levels between 0 and $l_1 - 1$ we do not count the root so we do not add the one, in the end we are left with $2(l_2 - 1 - l_1)$, added to the number of vertices on l_1 and l_2 we have $L(l_1)+l(l_2)+max\{0, 2(l_2-l_1-1)\}$ as the number of vertices in C. Thus the lemma is true for all the cases.

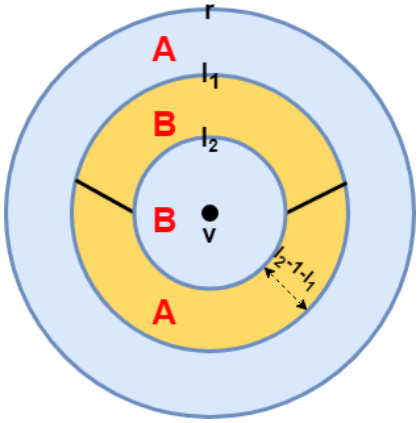

Figure 2. A possible case for the separation using this lemma is shown on this picture. A would receive the bigger part of those between levels 0 through $l_1 - 1$ and $l_2 + 1$ through r as well as the bigger of the components left after the separation of the middle part (which in this case was with bigger cost than $2/3$ of the whole graph). B would be assigned all other vertices except the ones that are exactly on the layers l_1 and l_2 as well as on the separator of the middle part C^*, because these would go to C.

F. The Planar Separator Theorem [5]

Theorem 4. *Given any n-vertex planar graph G (here the graph does no longer have to be connected like in lemma 2).*

The vertices of G again have non-negative costs, whose sum is not exceeding one.

The vertices of G can be separated into the sets A, B and C in such a way that vertices in A and B do not have shared edges, A and B both have costs under $2/3$ and C contains no more than $2\sqrt{2}\sqrt{n}$ number of vertices. Once proved this theorem can be used in order to separate any planar graph which is in fact commonly used to solve problems efficiently as we are going to see later on.

Proof: For this proof again we have to observe the different possibilities for graphs the theorem can be applied on and prove each one separately.

Firstly we are going to take a look at connected graphs. If G is connected than the lemma is obviously true, because we only have to apply the layering strategy and use lemma 2. This will result in a suitable separation as we have already proved. In order to estimate how many vertices there would be in C we first have to find the suitable layers for lemma 2. Firstly we are surching for such a level l_1, so that the sum of the costs on levels between 0 and $l_1 - 1$ is less than $1/2$ and between 0 and l_1 at least $1/2$. We will refer to the number of vertices on levels 0 through l_1 as k. Once we have found l_1 we also have to find a level l_0, which is closer to the root v than l_1 is. According to [3] every planar graph on n vertices has treewidth of $\mathcal{O}(\sqrt{n})$, so $L(l_0) + 2(l_1 - l_0) \leq 2\sqrt{k}$ should apply. After that we search for the layer l_2, which lays further from the root than l_1 does. For l_2 has to apply $L(l_2) + 2(l_2 - l_1 - 1) \leq 2\sqrt{n-k}$. If such layers exist than after the separation C would have $2(\sqrt{k} + \sqrt{n-k}) \leq 2(2\sqrt{n/2} = 2\sqrt{2}\sqrt{n}$ number of vertices.

The second case is when G is not connected. This would mean that there are k components of the graph for $k \in \mathbb{N}$. We are going to call these components respectively G_1, G_2, G_3, ...,G_k and their vertex sets V_1, V_2, V_3, ..., V_k. Now we have to observe the costs these components have and prove the theorem accordingly.

If there is no connected component which exceeds the cost of $1/3$ then we can simply take a union of some of theirs sets of vertices. We have to find the minimal possible index $i \in \mathbb{N}$ for which applies $V_1 \cup V_2 \cup ... \cup V_i$'s cost is more than $1/3$ then we can assign $V_1 \cup V_2 \cup ... \cup V_i$ to A, $V_{i+1} \cup V_{i+2} \cup ... \cup V_k$ to B and C would be left to contain no vertices. In such a partition all the rules of the theorem apply so it is proven to be true.

In the case when there is a component (G_i) with a cost between $1/3$ and $2/3$ we can assign all its vertices (the set V_i) to A, all other vertices to B and C would be assigned no vertices. All of the guidelines have obviously been satisfied so here the theorem also applies.

The last case that is left is when there is a component, whose cost exceeds $2/3$. Here we again can use lemma 2 to partition its vertices into the suitable sets A^*, B^* and C^*. After that we A can be assigned either A^* or B^*, whichever has greater cost, C would take the vertices in C^* and B would be left with the rest. We can easily calculate the worst case in order to see, if every rule of the theorem is satisfied. The lowest cost that A can have in the end would be when there is a component, which has cost of $2/3$, and after the separation $1/2$ of it are assigned to A, so we get $\frac{2}{3} * \frac{1}{2} = \frac{2}{6}$, which is less than $\frac{4}{6} = \frac{2}{3}$. In this worst case B would get less than $\frac{4}{6}$. For the number of vertices in C we can use the same logic as in the first case we have observed and prove it that way.

IV. An algorithm for finding a separation efficiently

We are now going to take a look at an algorithm [5] which is based on the proof of Theorem 4. This algorithm has a linear runtime $\mathcal{O}(n)$ as we are going to see.

Step 1: The first step is to embed the graph in the plane. Once we have done that we can store the information about the embedding in the following data structure. For each of the edges we are storing four pointers to incident edges positioned immediately clockwise and counter-clockwise to the endpoints of the described edge. For each of the vertices an incident edge is being stored. There is a way for this step to be done in $\mathcal{O}(n)$ described in [3].

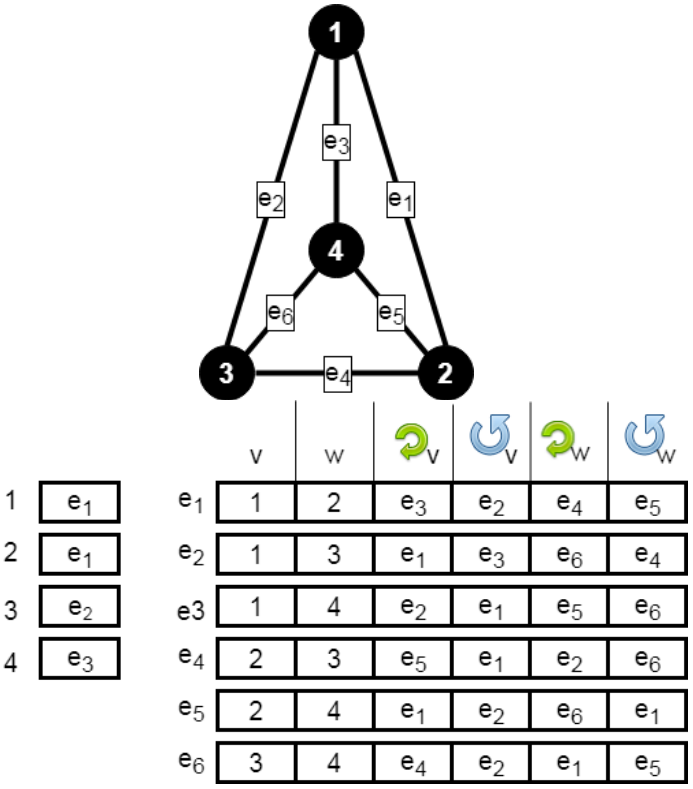

Figure 3. The above drawn graph is mapped in the data structure we have described already. For every vertex one of its incident edges is stored and for every edge the incident edges to its endpoints clockwise and counterclockwise are also stored.

Step 2: Now we need to find which are all the connected components. We scan, if there are exists a component which has a cost bigger than $2/3$. If no such component is found, then we can partition the vertices as described in the proof of Theorem 4. Otherwise we have to proceed to the next step. This calculations run at $\mathcal{O}(n)$.

Step 3: A spanning tree of the most costly component is needed and we acquire it by executing a breadth-first search from the root vertex v, that has been chosen before the algorithm started. We store the level of each vertex as well as the number of vertices on each level ($L(l_i)$ for $i \in [0, r]$). This step can also be done in $\mathcal{O}(n)$.

Step 4: In this step we are searching for the level l_1, that was described in the proof of Theorem 4. The summed cost of the levels from 0 through $l_1 - 1$ has to be no more than $1/2$ and the cost from 0 through l_1 has to be at least $1/2$. Again we are using k to refer to the number of vertices on levels between 0 and l_1. With the information from Step 3 this step also can be done in $\mathcal{O}(n)$.

Step 5: Using l_1 we find the highest level $l_0 \leq l_1$ and the lowest $l_2 \geq l_1$. Where $L(l_0) + 2(l_1 - l_0) \leq 2\sqrt{k}$ and $L(l_2) + 2(l_2 - l_1 - 1) \leq 2\sqrt{n-k}$ have to apply. Finding these two levels can be done in $\mathcal{O}(n)$.

Step 6: In order to simplify our graph we delete all vertices on level l_2 and above and shrink all vertices on levels 0 through l_0 into a single vertex x. The shrinking can be done by constructing a boolean table which contains the information for every vertex on levels 0 through $l_2 - 1$, if it is on level 0 through l_0 or not. We scan the edges incident to the subtree that was calculated for levels between 0 and l_0 and either delete the clockwise incident edge to the tree, if it has value true in the table or exchange it with one incident to x if its value is false. After this step we can apply lemma 2 on the shrunken graph. The step is done in $\mathcal{O}(n)$.

Step 7: We calculate a spanning tree of the newly formed graph by doing a breadth-first search, which can be done based on the previously calculated spanning tree for the whole graph to spare runtime. We store information about the parents of each vertex as well as the total cost of its descendants in order to simplify further work with this structure. Finally in this step we triangulate the graph by adding new edges, where they are needed. Here the time is also $\mathcal{O}(n)$.

Step 8: Then we choose an edge (v_1, w_1), either arbitrarily or in some preprogrammed way that compliments the specific graph that we are working with, so that the optimization in the next step are done faster or are not needed at all. We follow the parents of the endpoints of the chosen edge up to the root using the parents that were stored in the previous step. Then we calculate the costs of the sectors that can be distinguished according to the formed cycle. The side of the cycle with bigger cost is chosen to be the inside and is referred to as such later on. Using the data structure we have established in step 7 this step takes linear time as well.

Step 9: Now that we have chosen the cycle we are going to work with, we have to optimize it in order for it to satisfy the rules of theorem 4. This can be done by iterations of choosing more suitable cycles and then checking if they satisfy the theorem. Let us call the currently analyzed non-tree edge (v_i, w_i). Firstly we check, if the cost inside this cycle is greater than $2/3$, if that is the case we have to find a better candidate.

Here we have two different cases in which we have to continue accordingly with the calculations. Firstly we choose a triangle (v_i, y, w_i) where y is a vertex inside the (v_i, w_i) cycle. If one of the edges (v_i, y) or (y, w_i) is a tree edge and the other is not, then we chose (v_{i+1}, w_{i+1}) to be the non-tree edge of the two. After that we can recalculate the cost inside the new cycle, using the cost inside the (v_i, w_i) cycle and the costs of v_i, w_i and y respectively.

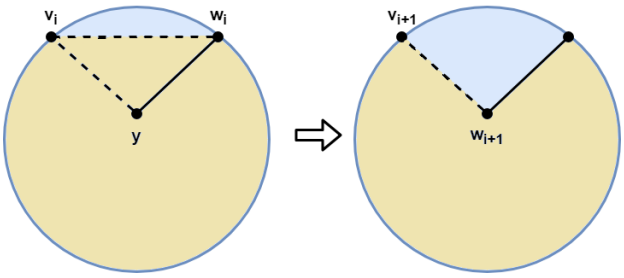

Figure 4. This diagram shows the case when one of the edges (v_i, y) or (y, w_i) is a non-tree edge (in this case (v_i, y). The area marked with yellow is the part of the graph that is in the cycle we are currently checking. After the step the only changes thing is that the edge (v_i, w_i) is no longer a part of the cycle. By repeatedly shrinking the size of the cycle in such way we get a suitable one.

If neither of the two edges (v_i, y) and (y, w_i) is a tree edge, then we firstly have to find the tree path from the edges v_i and w_i to y, by following their parents. Once we have done that we calculate the costs inside the (v_i, y) and (y, w_i) cycles. We chose (v_{i+1}, w_{i+1}) to be the edge which has the greater cost inside of its cycle.

We repeat the steps described in this step until we have found a cycle, whose inside cost is no more than $2/3$. Because we delete at least one face with every iteration we can conclude that this step also runs in $\mathcal{O}(n)$.

Step 10: Now that we have calculated all the structures that we need like the levels and the the cycle, we can finally partition the vertices using the techniques we have seen in the proofs of the lemmas and theorems we have already seen. Because we already have the data structures needed at our disposal this step can be terminated in $\mathcal{O}(n)$ time.

Thus we are done with the algorithm and the result is a partitioning of G's vertices in the sets A, B and C, which satisfy the parameters for a separator described in theorem 4.

V. Applications of the planar separator theorem

Now that we have seen how the algorithm works and what its costs are, we are going to discuss some of its usages. There are a lot of problems that can be solved significantly faster by using the divide and conquer strategy. There are also some other interesting applications and we are going to mention some of them.

A. Approximating NP-complete problems (Maximum independent set) [6]

Many NP-complete problems can be approximated using theorem 4 and the algorithm, we have described above. This can help solve the problem significantly faster but often at the cost of the answers accuracy. In many situations we have

to choose, if we want to sacrifice runtime for more precise solution. One such problem is the maximum independence set problem. We are given an n-vertex graph G and the aim is to find such a set of vertices, that no two vertices within it are connected by a shared edge. In order to solve it with the divide and conquer strategy the problem has to be simplified, so in the end we would not get the precise solution to the maximum independent set problem but rather an approximation for it. Instead of searching for vertices, that are not connected, we are going to find connected components of a suitable for us size, which have no edges, that connect the different components with each other, by recursively dividing the graph.

Theorem 5. *Given a planar graph G with non-negative vertex costs summing to no more than one and given a constant $0 \leq \varepsilon \leq 1$, we can find a set C, which contains $\mathcal{O}(\sqrt{n/\varepsilon})$ vertices, whose removal would leave the graph with no connected components of cost exceeding ε. Furthermore this set can be found in time $\mathcal{O}(n \log n)$.*

Proof: If $\varepsilon \leq \frac{1}{\sqrt{n}}$, then the theorem applies by taking the whole graph G. If that is not the case than we have to execute the following recursive algorithm.

Initialization: In the beginning we initialize the algorithm with an empty set of vertices in the separator C ($C = \emptyset$).

General step: As long as there can be found any components with cost bigger than ε we choose one such component K and apply Theorem 4 to it which separates it in three sets as described in Section 3. After that we unite the sets of the C we had before the step and the separator of K that we have found. We also assign a level to every component using the following pattern. The levels which exist when the algorithm terminates are assigned level 0 and every component, which was separated, gets a level equal to the maximum level of the components, it was split in, plus one. Using this approach we can ensure that any two components on the same level do not have shared edges between each other. The algorithm looks as follows in pseudo code:

```
1 while (∃K ∈ G(V − C, E) with cost(K) > ε) {
2     apply Theorem 1 to K resulting in: {
3         A_1 with cost(A_1) ≤ 2/3cost(K),
4         B_1 with cost(B_1) ≤ 2/3cost(K),
5         number of vertices in C_1 ≤ 2√2√n_k
6         }
7     C = C ∪ C_1;
8     level(K) = level(max(level(A_1), level(B_1)) + 1;
9 }
```

Analysis: Each components with cost at least one has a cost bigger than ε, because it was obviously split by the algorithm so it satisfied the conditions of the while loop. We would have costs for the levels of such pattern:

1) At level 1 we would have cost greater than ε
2) At level 2 the cost would be greater than $3/2\varepsilon$
3) At level 3 it would be at least $(3/2)^2\varepsilon$

$\vdots$

So we can conclude that each component on level $i \geq 1$ the cost would be at least $(\frac{2}{3})^{i-1}\varepsilon$. Because the total cost of G is always at most 1 as stated in the theorem we can calculate the number of vertices on each level as

$$N_i \leq \frac{1}{(3/2)^{i-1}}\varepsilon = \frac{(2/3)^{i-1}}{\varepsilon}$$

Since we know that $0 \leq \varepsilon \leq 1$ for the maximum level k it applies that

$$1 \leq \frac{(2/3)^{k-1}}{\varepsilon} \leq (2/3)^{k-1}\sqrt{n}$$

That means that $k \leq (\log_{3/2} n)/2+1$, or put in other words we have a logarithmic rising number of separations in comparison to the number of vertices. So the algorithm has $\mathcal{O}(\log n)$ steps. We know that a separation happens in linear time so the whole algorithm runs in time $\mathcal{O}(n \log n)$.

Now we have to calculate the size of C in order to prove the whole theorem. We are going to refer to the components on some level $i \geq 1$ as $K_1, K_2, ..., K_l$ of size $n_1, n_2, ..., n_l$ respectively. The number of vertices that are added to C with every separation is bounded by $2\sqrt{2}\sum_{j=1}^{l}\sqrt{n_j}$. As we have already seen that $l \leq (2/3)^{i-1}/\varepsilon$ and we know that $\sum_{j=1}^{l} n_j \leq n$, it can be concluded that $\sum_{j=1}^{l}\sqrt{n_j}$ is maximized when we set $n_j = n/l$ for $1 \leq j \leq l$. So we get

$$2\sqrt{2}\sum_{j=1}^{l}\sqrt{n_j} \leq 2\sqrt{2}\sqrt{nl} \leq 2\sqrt{2}\sqrt{n/\varepsilon(2/3)^{(i-1)/2}}$$

So the size of C can be estimated as

$$\sum_{i=1}^{\infty} 2\sqrt{2}\sqrt{n/\varepsilon(2/3)^{(i-1)/2}} = \mathcal{O}(\sqrt{n/\varepsilon})$$

.

With that we have proved the theorem and can now use it in order to approximate the maximum independent set problem.

The algorithm we are going to use needs a function $k(n)$ to be defined later on and has the following structure:

1) Step 1: We apply Theorem 4 to G and set $\varepsilon = k(n)/n$ and each vertex's cost to $1/n$. In this way we get the separator set C, which is no bigger than $\mathcal{O}(n/\sqrt{k(n)})$.
2) We search for a maximum independence set in every connected component after the separation. The maximum independent set I for the whole graph is then formed as a union of all the maximum independent sets from each connected component.

The first step runs in $\mathcal{O}(n \log n)$ as we have proved already in Theorem 5. Because we have to check for every subset of every component to see whether there exists an independent set, we get a runtime of $\mathcal{O}(n_i 2^{n_j})$ for checking a component with n_i vertices. The total time for step 2 to terminate is then

$$\begin{aligned}\mathcal{O}(max\{\sum_{i=1}^{n} n_i 2^{n_j} | \sum_{i=1}^{n} n_i = n \text{ and } O \leq n_i \leq k(n)\}) \\ = \mathcal{O}(\frac{n}{k(n)}k(n)2^{k(n)}) \\ = \mathcal{O}(n2^{k(n)}).\end{aligned}$$

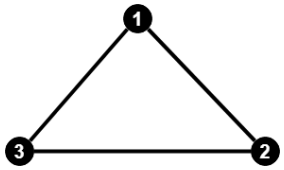

Figure 5. Even for a graph as simple as a triangle the set of subsets of vertices includes: $\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$. So the number of subsets is $8 = 2^3$.

So the entire algorithm takes $\mathcal{O}(n \cdot max\{\log n, 2^{k(n)}\})$ time. But as we mentioned earlier this is not the exact solution to the problem but rather an approximation, which means that we should expect an error. Now we are going to analyze the impact of the error and give a suggested value for the function $k(n)$, which offers a balance between the runtime of the algorithm and its error.

Let us assuming that I^* is the maximum independent set of G. its restriction to one of the components formed after the separation step is surely lesser than the restriction of I to the same component. From that follows that $|I^*| - |I| = \mathcal{O}n/\sqrt{k(n)})$. The planarity of G means that it is also four-colorable [2], so $|I^*| \geq n/4$. This means that the relative error is $(|I^*| - |I|)/|I^*| = \mathcal{O}(1/k(n))$. The conclusion is that the relative error tends to zero when the number of vertices in G razes. Running the algorithm on larger graphs would also require more steps so the runtime is getting longer.

If we let $k(n) = \log n$ the algorithm would take $\mathcal{O}(n^2)$ time algorithm which produces a relative error which does not exceed $\mathcal{O}(1/\sqrt{\log n})$. Another option would be to set $k(n) = \log\log n$, which would speed up the algorithm to $\mathcal{O}(n \log n)$ time and $\mathcal{O}(1/\sqrt{\log\log n})$ relative error.

As we have shown a polynomial time can be achieved at the cost of an error in the calculations. Using this approach to solving the maximum independent set problem also requires careful planning and setting the parameters of the algorithm to the most suitable values in order to archive balance between speed and accuracy.

B. Embedding of data structures [6]

Another interesting application of the planar separator theorem can be found in the case when we want to represent one data structure using another. We want to preserve the information about the adjacency of two nodes in the first structure by placing them close to each other in the second. Before talking more about this problem we have to introduce some more notations.

An embedding of an undirected graph $G_1 = (V_1, E_1)$ in another undirected graph $G_2 = (V_2, E_2)$ is a one-to-one map $\phi : V_1 \to V_2$, in other words ϕ is a bijective function which projects every node from V_1 in exactly one of the nodes in V_2 and vice versa. The worse case proximity of this function has to be at most $\{d_2(\phi(v), \phi(w))|\{v, w\} \in E_1\}$, where $d_2(x, y)$ represents the distance between x and y in the second graph G_2. The average case would be $(1/|E_1|)/\sum\{d_2(\phi(v), \phi(w))|\{v, w\} \in E_1\}$. So to sum things up the problem is to make such an embedding for G_1 that the worst case proximity is minimized.

Theorem 6. *Any planar graph with maximum degree k can be embedded in a binary tree so that the average proximity is $\mathcal{O}(k)$*

Proof: This problem can also be solved recursively. The algorithm, we are going to apply, works as follows. If G contains only one vertex v, then the tree T that it is embedded in also contains one vertex, which is the image of v. Otherwise if that is not the case, we apply theorem 4 with vertex costs of $1/n$ on every vertex in G. Once we have acquired the partition in the sets A, B and C we choose one of the vertices laying on the separator C (if the set of the separator is empty then we choose one of the vertices in A), we will refer to the chosen vertex as v. We embed the subgraph which includes only the vertices in $A \cup C - \{v\}$ in the tree T_1 recursively. The part of G that is induced by the set B is then being embedded in a binary tree T_2 also by recalling the algorithm recursively. Finally we create the tree T by setting v as its root and the two children it has would be the root of T_1 and the root of T_2. T obviously consists of exactly n vertices.

The maximum depth of a tree T with n vertices, which was produced in the way we have described, would be referred to as $h(n)$. Then it applies that

$$h(n) \leq h(2n/3 + 2\sqrt{2}\sqrt{n} - 1) + 1 \leq h(29n/30) + 1$$

That proves that the height of the binary tree we have induced is $\mathcal{O}(\log n)$.

If we let G_1 be the subset of G with the vertices from $A \cup C$ and G_2 to be the one induced by the set B, we can define a function $s(G) = \sum\{d_2(\phi(v), \phi(w))|\{v, w\} \in E\}$ in order to represent the proximity. This function can also be displayed as follows

$$s(G) \leq \begin{cases} 0, & \text{for } n = 1 \\ s(G_1) + s(G_2) + 2k|C|h(n), & \text{for } n > 1 \end{cases}$$

It is so, because the edges that were in G but were not derived in neither of the graphs G_1 and G_2 were incident to a vertex in C. So if we call $s(n)$ the maximum value of $s(G)$ for the n-vertex graph G, then

$$\text{s(n)} \leq max\{s(i) + s(n - i - 1) + ck\sqrt{n}\log n | n/3 - 2\sqrt{2}\sqrt{n} \leq i \leq 2n/3 + 2\sqrt{2}\sqrt{n}\}$$

if $n > 1$for some positive constant c

We can conclude that $s(n)$ is in $\mathcal{O}(kn)$.

In order to say if the proximity really is $\mathcal{O}(k)$ we first have to observe the cases for a connected and non-connected graph G. For a connected n-vertex graph and it was embedded using the algorithm we have described, then G contains at least $n - 1$ edges, and the average proximity would be $\mathcal{O}(k)$. If G is not connected on the other side then by embedding each

component and combining the results would get us an average proximity again of $\mathcal{O}(k)$. So we have confirmed the theorem.

VI. Conclusion

As we have seen the planar separator theorem can be very useful in many algorithmic approaches. There are some data structures that are very helpful by applying the theorem like a spanning tree in the connected components, whose vertices we wish to partition, the leveling system we have shown in Lemma 2, which can speed up the process of finding the suitable separator significantly and other ones, which can narrow the variables in the calculations and in this way speed them up, but also limit the targeted graph set, the algorithm can be successfully applied on.

Separating a problem to smaller ones of the same kind has proven over time to be a very effective method for solving complex problems in applicable times. There are many examples of that and we have also shown in the paper that NP-complete problems can be sped up to polynomial times at the cost of the error, that is caused by removing the vertices in the separator set form the graph. Often times an approximation of the exact solution for a problem is useful enough for one to make conclusions about the certain situation he is researching and the divide and conquer approach is widely spread option for finding such approximations.

Recursion is also a technique that works very well in combination with the divide and conquer strategy. Often times the vertices in the separation set can be chosen as suitable elements for algorithms, as they, by definition, should lie around the center of the graph. By separating the graph and choosing such vertices recursively a lot of non deterministic decisions can be made comparably faster. We have shown that embedding one data structure in another without changing the semantics significantly is possible by using this exact approach.

References

[1] Guilford L. Spencer D. Wick Hall. Elementary Topology. *Bulletin of the American Mathematical Society*, 1955.

[2] Peter Dörre. Every planar graph is 4-colourable – a proof without computer. 8.

[3] John Hopcroft and Robert Tarjan. Efficient Planarity Testing. *Journal of the ACM*, 21(4):549–568, oct 1974.

[4] Casimir Kuratowski. Sur les espaces complets. *Fundamenta Mathematicae*, 15(1):301–309, 1930.

[5] Richard J. Lipton and Robert Endre Tarjan. A Separator Theorem for Planar Graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, apr 1979.

[6] Richard J. Lipton and Robert Endre Tarjan. Applications of a Planar Separator Theorem. *SIAM Journal on Computing*, 9(3):615–627, aug 1980.

[7] Dániel Marx. The Square Root Phenomenon in Planar Graphs. pages 28–28. Springer, Berlin, Heidelberg, 2013.