A Linear Time Algorithm for Testing if G has a K_5 Minor

B. Reed and Z. Li, McGill University

THIS IS A DRAFT WHICH IS NOT COMPLETELY POLISHED. ALSO WE HAVE NOT DONE A COMPLETE LITERATURE SEARCH SOME OF OUR SUBROUTINES MAY HAVE BEEN DEVELOPED BY OTHERS PREVIOUSLY.

1 Introduction

The graph K_ℓ is a graph on ℓ vertices with all possible edges between them. A graph $G = (V(G), E(G))$ is said to have (or contain) a K_ℓ minor if we can obtain K_ℓ from G via a sequence of edge contractions and vertex deletions. A K_ℓ-model of G is a set of ℓ disjoint connected subgraphs $\{X_i\}_{i=1}^{\ell}$ such that for all distinct i, j, there exists an edge $xy \in E(G)$ with $x \in X_i, y \in X_j$. In this case, xy is called an *edge between* X_i and X_j. It is easy to show that G has a K_ℓ minor if and only if G has a K_ℓ-model. All graphs are undirected and simple unless otherwise stated.

We present a linear time algorithm which determines whether a graph G has a K_5 minor and finds a K_5-model if it exists. We start by describing a polynomial time algorithm which does this and then improve it to a linear time algorithm.

1.1 A simple algorithm

To warm up, we describe a simple recursive algorithm which solves the problem in polynomial, but not linear time.

Clearly, G has a K_5-model if and only if one of its connected components has a K_5-model. Therefore, if the graph is not connected, we can recurse, running our algorithm on each of its connected components seperately.

In fact, this idea can also be applied when G has a cut vertex v. Suppose G has a K_5-model $\{X_i\}_{i=1}^{5}$. Then at most one X_i, without loss of generality X_1, contains v. For $i \geq 2$, X_i is completely contained in some component of $G - v$. If X_i and X_j are contained in different components then there is no edge between them. Therefore, all the $X_i, i \geq 2$ must be contained in the same connected component U of $G - v$. Let C be the graph induced by the vertices $U \cup \{v\}$. Any component of $X_1 \cap C$ not containing v is a component of X_1, so $X_1 \cap C$ is connected as X_1 is. Furthermore, every edge between the X_i has one endpoint in U and hence both endpoints in C, so replacing each X_i by $X_i' = X_i \cap C$ gives a K_5-model $\{X_i'\}_{i=1}^{5}$ of G where $X_i' \subseteq C$ for all i.

Therefore, we can do the following: determine if G has a cut vertex and if so find such a vertex v, find the connected components $U_1, \ldots, U_k$ of $G - v$, and then recurse on $C_i, \ldots, C_k$ where C_i is the subgraph induced by $U_i \cup \{v\}$.

Now we only need to determine whether G contains a K_5 minor when G is 2-connected.

The same idea can be applied to cutsets of size 2. Let $X = \{u, v\}$ be a cut in a 2-connected graph G. Let $\{X_i\}_{i=1}^5$ be a K_5-model of G. Since the X_i are disjoint, at most two of them intersect X. All other X_i's are contained in the same connected component U of $G - X$. Let C be the graph obtained from the subgraph of G induced by $U \cup X$ by adding the edge uv if it was not already in G. Let $X_i' = X_i \cap C$. Every component of $X_i' \cap C$ which contains neither u nor v is a component of X_i. So, since uv is an edge of C, every X_i' is connected. If the edge of G between X_i and X_j is not in C then both X_i and X_j must intersect $G - U$ and hence one contains u and the other contains v and they are joined by the edge uv of C. Thus, $\{X_i'\}_{i=1}^5$ is a K_5-model in C.

Therefore, given a 2-connected input graph G, we can determine if G is 3-connected. If not, we find a cutset $X = \{u, v\}$ in G, find the connected components $U_1, \ldots, U_k$ of $G - X$. Let C_i be the subgraph induced by $U_i \cup \{u, v\}$ with an added edge uv and recurse on $C_i, \ldots, C_k$. Since G is 2-connected, each C_i is a minor of G, and so if any C_i has a K_5 minor so does G. By the above remarks, if G has a K_5 minor so does one of the C_i. Thus, solving these subproblems does lead to a solution for G.

Now we only need to be able to determine whether G contains a K_5 minor when G is 3-connected.

Our approach needs to be modified for cutsets of size 3. It is not true for an arbitrary cutset X of size three in a 3-connected graph G that G has no K_5 minor precisely if for every component U of $G - X$, the graph obtained from $G[X + U]$ by adding a clique on X has no K_5 minor. See Figure 1 for an example. The problem is that our auxiliary graphs need not be minors of G. However, if $G - X$ has at least three components, this fact is true. That is, we have:

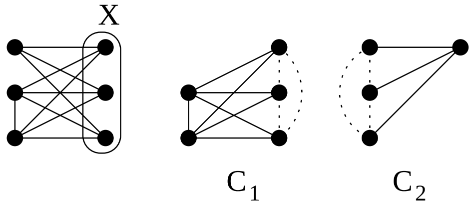


Figure 1: G is a $K_{3,3}$ with an added edge. X is a cut which splits G into C_1 and C_2. C_1 is K_5 but G has no K_5 minor

Observation 1.1. *Suppose $X = \{x, y, z\}$ is a 3-cut in a 3-connected graph G such that $G - X$ has at least 3 components. Then G has no K_5 minor precisely if for every component U of $G - X$, the graph C obtained from $G[X + U]$ by adding a clique on X has no K_5 minor.*

Using the same approach as in the previous two cases, we can easily show that if G contains a K_5 minor, some C contains a K_5 minor. Now we show that C is indeed a minor of G. We contract two of the other components of $G - X$ to a single vertex and delete all other components. Call the vertices resulting from the contraction a and b. Each of a, b has an edge to each vertex of X since G is 3-connected. Contract ax to add edges xy and xz.

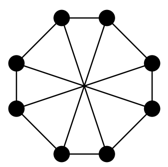
Contract by to add edges yx and yz. We obtain C since all edges of the triangle have been added.

For simplicity, we will call a cut of size i that splits G into $\geq j$ connected components an (i, j)-cut. Therefore, the cuts discussed in the observation are $(3, 3)$-cuts. We can repeatedly apply the observation to recurse, thereby reducing the problem to the case where the graph has no $(3, 3)$-cuts.

But these graphs are easy to handle due to Wagner's theorem for K_5 minor free graphs.

Fact 1.2. *If a 3-connected graph G has no K_5 minor then at least one of the following is true:*

1. *G is the graph L in the accompanying figure.*
2. *G is planar.*
3. *G has a $(3, 3)$-cut.*



L

We give an algorithmic proof of this fact in section 8.3. A reference to the original proof will be added when we find it.

Now we test if G is L and if G is planar. So if G has no $(3, 3)$-cut, is not L and is not planar, we know G has a K_5 minor (and this can be found in linear time using the algorithm described in section 8.3). Otherwise, since G has no $(3, 3)$-cut, G has no K_5 minor.

As shown below, standard first year undergraduate analysis shows that when we apply these recursions, to a graph with n vertices and m edges, we recurse at most $O(n)$ times and hence split our problem up into at most $O(n)$ subproblems the size of whose vertex and edge sets is $O(n + m)$. We can then solve each of these problems seperately, using a linear time planarity testing algorithm (since testing if a graph is L is trivial). Thus, the total running time for the second step is $O(n + m)$.

It remains to bound the complexity of the first step. There is a linear time algorithm to perform all the recursions which use cutsets of size one or two, the difficulty is the cuts of size three. Naive approaches yield $O(n + m)$ complexity per iteration. We will provide an $O(n + m)$ algorithm to completely decompose the graph.

In the next subsection, we present some data structures which we will use to keep track of our decompositions, and the pieces into which it decomposes G.

In the last subsection of this introductory chapter we give the high level description of our recursive linear time algorithm which either finds a K_5 model of our input graph or constructs such a decomposition and the corresponding data structures.

1.2 Complete Block Decompositions

Note that, in the previous section, we have reduced the problem of determining whether a graph G contains a K_5 minor to the problem of determining whether some graphs $G_1, \ldots, G_\ell$ are planar or isomorphic to L. We would get a faster algorithm if we could determine $G_1, \ldots, G_\ell$ directly (without having to recursively find cutsets in G).

The algorithm in the previous section can be summarized as follows.

1. Obtain the connected components $G_{1,1}, \ldots, G_{1,\ell_1}$ of G
2. Obtain the 2-connected graphs $G_{2,1,} \ldots, G_{2,\ell_2}$ by recursively finding cut vertices in $G_{1,1}, \ldots, G_{1,\ell_1}$.
3. Obtain the 3-connected graphs $G_{3,1}, \ldots, G_{3,\ell_3}$ by recursively finding cuts of size 2 in $G_{2,1}, \ldots, G_{2,\ell_1}$.
4. Obtain the 3-connected graphs with no $(3,3)$-cuts $G_{4,1}, \ldots, G_{4,\ell_4}$ by recursively finding $(3,3)$-cuts in $G_{3,1}, \ldots, G_{3,\ell_3}$.
5. Check each of $G_{4,1}, \ldots, G_{4,\ell_4}$ for planarity and isomorphism to L.

We can represent each decomposition that we construct via a tree. We now give the formal definitions of these trees. Our definitions differ slightly from previous ones since our graphs are simple and we need to keep track of the connected components (see [9, 7, 1] for other definitions).

A block tree is only defined for connected graphs.

Definition 1.3. A *block tree*, also called a *1-block tree*, of a connected graph G, written $[T, \mathcal{G}]$, is a 2-coloured tree T with a set $\mathcal{G} = \{G_t\}_{t \in T}$ with the following properties.

- G_t is a graph for each $t \in T$
- If G is 2-connected then T has a single node r which is coloured 1 and $G_r = G$.
- If G is not 2-connected then there exists a colour 2 node $t \in T$ such that
  1. $G_t = \{v\}$ and v is a cut vertex in G.
  2. Let $T_1, \ldots, T_k$ be the connected components (subtrees) of $T - t$. Then $G - v$ has k components $U_1, \ldots, U_k$ and there is a labelling of these components such that T_i is a block tree of $G_i = G[U_i \cup v]$.
  3. For each i, there exists exactly one colour 1 node $t_i \in T_i$ such that $v \in G_{t_i}$.
  4. For each i, $tt_i \in E(T)$.

A 2-block tree is only defined for 2-connected graphs.

Definition 1.4. A *2-block tree* of a 2-connected graph G, written $[T, \mathcal{G}]$, is a tree T with a set $\mathcal{G} = \{G_t\}_{t \in T}$ with the following properties.

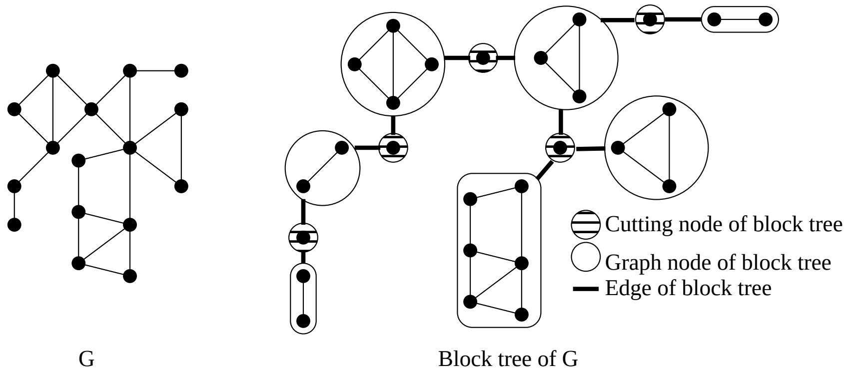


Figure 2: An example of a block tree. The G_t's are drawn inside the node t

- G_t is a graph for each $t \in T$
- If G is 3-connected then T has a single node r which is coloured 1 and $G_r = G$.
- If G is not 3-connected then there exists a colour 2 node $t \in T$ such that
  1. $G_t = \{u, v\}$ and $\{u, v\}$ is a cut in G.
  2. Let $T_1, \ldots, T_k$ be the connected components (subtrees) of $T - t$. Then $G - \{u, v\}$ has k components $U_1, \ldots, U_k$ and there is a labelling of these components such that T_i is a 2-block tree of $G_i = G[U_i \cup \{u, v\}] \cup \{uv\}$.
  3. For each i, there exists exactly one colour 1 node $t_i \in T_i$ such that $\{u, v\} \subseteq G_{t_i}$.
  4. For each i, $tt_i \in E(T)$.

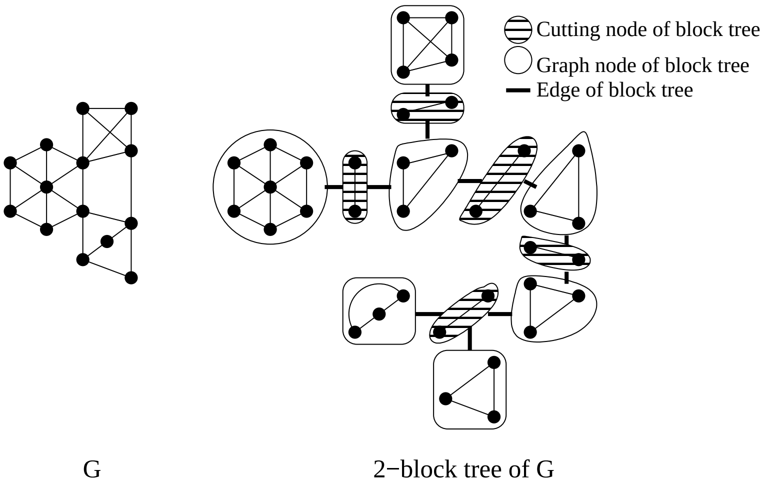


Figure 3: An example of a 2-block tree. The G_t's are drawn inside the node t

A $(3,3)$-block tree is only defined for 3-connected graphs.

Definition 1.5. A *(3,3)-block tree* of a 3-connected graph G, written $[T, \mathcal{G}]$, is a tree T with a set $\mathcal{G} = \{G_t\}_{t \in T}$ with the following properties.

- G_t is a graph for each $t \in T$
- If G has no $(3,3)$ cut then T has a single node r which is coloured 1 and $G_r = G$.
- If G has a $(3,3)$ cut then there exists a colour 2 node $t \in T$ such that
  1. $G_t = \{u, v, w\}$ and $\{u, v, w\}$ is a $(3,3)$-cut in G.
  2. Let $T_1, \ldots, T_k$ be the connected components (subtrees) of $T - t$. Then $G - \{u, v, w\}$ has k components $U_1, \ldots, U_k$ and there is a labelling of these components such that T_i is a 3-block tree of $G_i = G[U_i \cup \{u, v, w\}] \cup \{uv, vw, uw\}$.
  3. For each i, there exists exactly one colour 1 node $t_i \in T_i$ such that $\{u, v, w\} \subseteq G_{t_i}$.
  4. For each i, $tt_i \in E(T)$.

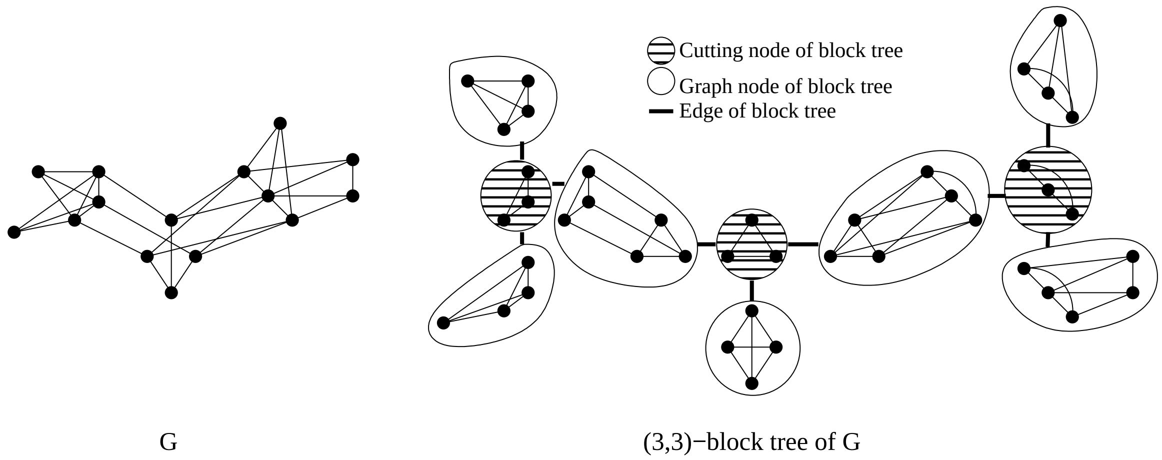


Figure 4: An example of a $(3,3)$-block tree. The G_t's are drawn inside the node t

Definition 1.6. A node that is coloured 1 (in a 1-block tree, 2-block tree or $(3,3)$-block tree) is called a *graph node* and a node that is coloured 2 is called a *cutting node.*

Definition 1.7. Let G be a connected graph. A *complete block decomposition (CBD)* of G is a block tree $[T, \mathcal{G}]$ and a 2-block tree $[S_t, \mathcal{G}_t]$ for each graph node $t \in T$ and a $(3,3)$-block tree $[R_{s_t}, \mathcal{G}_{s_t}]$ for each graph node $s_t \in S_t$, $t \in T$.

The graph nodes of the block tree are the $G_{2,i}$'s. The graph nodes of the 2-block trees are the $G_{3,i}$'s. The graph nodes of the $(3,3)$-block trees are the $G_{4,i}$. Therefore, we wish to find a CBD of G in linear time.

In doing so, we will use the fact proven below that every 3-connected graph G which is not $K_{3,3}$ has a unique $(3,3)$-block tree and this uses all the $(3,3)$ cuts of G. Thus, we may speak of THE $(3,3)$-block tree, as long as we avoid $K_{3,3}$ which we shall as we handle small graphs separately by brute force.

1.3 Complete block decomposition algorithm

We assume that the input graph is 3-connected as otherwise, we use the following algorithms to find a partial CBD consisting of the 1-block tree and 2-block trees for the graph nodes of the 1-block tree and recurse on the graph nodes of the 2-block tree.

Fact 1.8. *[8, 4, 2] There exists a linear time algorithm which*

1. *If G is connected, builds the block tree of G.*
2. *If G is 2-connected, builds the 2-block tree of G.*

We describe a recursive linear time algorithm which either finds a K_5-model or builds the $(3,3)$-block tree for G. In each iteration it either finds a K_5-model or recursively applies itself on a set of smaller 3-connected graphs. Having solved the problem on these smaller graphs, it uses the solution to these new problems to construct the solution to the old problem quickly. Our reductions will reduce the problem so much that provided the reduction and construction subroutines take linear time, then so does the entire algorithm.

Some of our reductions are straightforward. For example suppose the graph has a large set of vertices of degree 3 each of whose neighbourhood is the same as at least 4 vertices in G which are not in the set. In this case, deleting these vertices does not change the structure of the $(3,3)$ block tree. Indeed having constructed a tree for the smaller graph, to obtain a tree for G we note that the neighbourhood of each deleted vertex will be a $(3,3)$-cut in the tree, and for each such vertex v, we need to add a pendant graph node which is a clique on $v + N(v)$ off the unique cutting node containing $N(v)$. Note that the graph we reduce to is 3-connected. In section 8.1, we give a linear time algorithm which removes such a set of degree 3 vertices.

Another simple reduction involves finding as a subgraph of our input graph, a minor (indeed a subdivision) of a graph with high density. If, at any step of the algorithm, we find a minor of the input graph whose ratio of edges to vertices is greater than 64, we simply apply the following fact to find a K_5-model.

Fact 1.9. *[6] Let G be a graph. If $|E(G)| \geq 64|V(G)|$ then G has a K_5 minor and there exists a linear time algorithm which find a K_5-model of G.*

Another reduction involves finding a set of 3-cuts in G which decompose G into more than εn pieces. We then decompose our graph into smaller pieces using these cuts. We note that these cuts need not be $(3,3)$-cuts but they will be such that (a) given a K_5-model in one of the smaller graphs we can quickly find one in G and (b) we can paste $(3,3)$-block trees of the smaller graphs together into a $(3,3)$-block tree for G in linear time.

Our last reduction involves contracting a matching M in G with a linear number of vertices to obtain a new 3-connected graph H. Clearly if H has a K_5 minor so does G and we can uncontract to find this minor quickly. Obtaining our output for G given a $(3,3)$-block tree for H is much more complicated and forms the bulk of the paper. To do so, we need to insist that M has some special properties. To wit, it must use only vertices whose degree in G is small, and it must be induced.

Having described the algorithm informally we are now ready to present a formal specification.

Algorithm 1.10. We will use constants $d = 100,000, \varepsilon = d^{-6}$. The input is a 3-connected graph G. This algorithm returns either a K_5 model of G or a $(3,3)$-block tree of G. If G has fewer than 20 vertices we solve the algorithm by brute force, otherwise we proceed as follows.

1. Count the number of vertices in G. Count the number of edges in G. Stop counting if we reach $64|V(G)|$ and apply Fact 1.9 to find a K_5 minor in G and return it.

2. Sort the neighbourhoods of the vertices of degree 3 to find the ones that share a common neighbourhood with at least 4 others. Construct the graph H obtained by removing most of these degree 3 vertices leaving 4 vertices per neighbourhood. Call the resulting graph H. The algorithm to build H is described in more detail in section 8.1.

   If $|V(H)| < (1-\varepsilon)|V(G)|$ then go to Step 3 otherwise go to Step 4.

3. Recurse on H which is 3-connected and is not $K_{3,3}$.

   If a K_5 model of H is returned, return it as a K_5 model of G. If a $(3,3)$-block tree of H is returned, add back the degree 3 vertices that we removed to obtain the $(3,3)$-block tree of G. To add a vertex v, create a graph node t with $G_t = \{v\} \cup N(v)$. $N(v)$ is a $(3,3)$-cut in H. This guarantees the existence of a cutting node c in the $(3,3)$-block tree of H with $G_c = N(v)$. Add an edge tc to the $(3,3)$-block tree of H which contains c. The algorithm to add these nodes in linear time is described in section 8.2.

   The above procedure gives us a $(3,3)$-block tree of G. Return it.

4. Run the algorithm described in section 8.4 and then apply the procedure of Section 6 if it returns a matching. Since we are not in Step 3, this combined procedure either returns a K_5-model of G, a set of εn 3-cuts of G each of which contains an edge and none of which separates another, or an induced matching M of size $> \varepsilon|V(G)|$ amongst the vertices of degree at most d, such that contracting the edges of M yields a 3-connected graph.

   If a K_5-model of G is returned, return it.

   If a set of cuts is returned, decompose G into a set of εn smaller problems whose solutions can be pasted together (as we describe in section 7) to solve the original problem on G.

If a matching M is returned, contract the edges of M to obtain a graph H. Recurse on H. If a K_5-model of H is returned, return its uncontraction. If a $(3,3)$-block tree of H is returned, it is more complicated to obtain our output for G. The algorithm to do this is described in sections 4 and 5. We run this algorithm to obtain either a $(3,3)$-block tree of G or a K_5-model of G which we return.

As we remarked the most difficult step of the algorithm is Step 4. In sections 4, 5 and 8.4, we describe the subroutines that handles this case. With this subroutine in hand, we go on to complete the analysis of our algoirthm in section 8.6. Before doing so, we analyze how the small cutsets in G and H can interact in sections 2 and 3.

To close this section, we give a brief sketch of the key subroutine.

We want to massage the $(3,3)$-block tree of H to obtain a $(3,3)$-block tree of G (or a K_5-model of G). We look at where the $(3,3)$ cuts of G can be with respect to the decomposition of H. It turns out that they either correspond to the cuts in the $(3,3)$-block tree for H or correspond to a triangle of H or to certain P_3s of H which interact with H in a very special way. This allows us to refine our decomposition tree so that it actually contains a cutting node corresponding to every possible $(3,3)$-cut of G. We then process this tree, modifying it so as to obtain a $(3,3)$-block tree for G unless a K_5-model of G is found in the process.

2 Cutset structure

There is a unique 1-block tree for any graph. There is a unique $(3,3)$-block tree for any 3-connected graph except $K_{3,3}$ which has two $(3,3)$-block trees, these two trees both have only one cutting node, and indeed are isomorphic, differing only in the choice of which of the sides of the bipartition appears in the cutting node. In contrast, a 2-connected graph may have many different 2-block trees. Indeed if G is a cycle, for any triangulation of the cycle there is a 2-block tree whose cutting nodes correspond to the chords in the triangulation, and all of these trees are different. As we describe below, however, this is essentially the only ambiguity which can arise in the 2-block tree.

In this section, we prove the uniqueness of 1-bock trees and $(3,3)$-block trees, and examine what we can say about the features which all of the 2-block trees of a given 2-connected graph share.

We begin with the following simple lemma.

Lemma 2.1. *Suppose U is a component of $G-X$ for some minimal cutset X of G and Y is a cutset in the graph C obtained from $G[U \cup X]$ by adding edges so that X is a clique. Then Y is also a cutset of G, and the components of $C-Y$ disjoint from X are components of $G-Y$. If Y contains X then the components of $G-X-U$ are also components of $G-Y$, otherwise the union of $G-X-U$ and the component of $C-Y$ containing $X-Y$ forms one component of $G-Y$.*

Proof. A component of $C-Y$ disjoint from X has no edges to $G-C$ and is a subgraph of G so is indeed a component of $G-Y$. Since X is a clique in C it follows that Y is indeed a cutset in G. If Y contains X then obviously the components of $G - X$ disjoint from U (and hence Y) are components of $G - Y$. Finally the minimality of X implies that every component of $G - X$ has an edge to every vertex of X from which the last statement of the lemma follows. □

This lemma tells us that when we perform one step of our decomposition, we do not construct new cuts in the pieces into which we decompose the graph. I.e, every cutting node of one of our trees corresponds to a cutset of G. It also tells us that in every tree in which it is used, the cutset decomposes G into the same pieces (the reader may observe this phenomena in the triangulation example mentioned earlier). As the triangulation examples shows however, we may destroy cutsets when we add an edge corresponding to a 2-cut. We now examine which cuts can be destroyed and which cannot.

We say a cutset Y seperates X if there are vertices of X in different components of Y. If decomposing on X is to change the component structure with respect to Y then clearly Y must seperate X. We now examine for which pairs of cuts this can actually happen. It clearly cannot happen if X has only one vertex. This fact explains the uniqueness of the 1-block trees. The following lemma explains the uniqueness of the $(3,3)$-block trees.

Lemma 2.2. *Let U be a 3-connected graph that is not $K_{3,3}$. Let X be a $(3,3)$-cut in U or a $(3,2)$-cut such that $U[X]$ is connected. There does not exist a $(3,3)$-cut separating the vertices of X.*

Proof. Suppose the lemma is false. Let $X = \{a, b, c\}$. Let $U_1, \ldots, U_k$ be the connected components of $U - X$. Let Y be a $(3,3)$-cut separating X. WLOG, Y separates $\{a, b\}$.

Y must contain a vertex of U_1 and a vertex of U_2. Hence, every component of $U - Y$ which contains a neighbour of every vertex of Y must intersect X. Since there are three such components, we see that the vertices of X are in different components of $U - Y$. This implies that $U[X]$ is not connected and hence $k \geq 3$. Since $U[U_3 \cup X]$ is connected, Y contains a vertex of U_3. Thus, X and Y are disjoint and the vertices of Y are in different components of $U - X$. Thus, each component of $U - X - Y$ has edges to at most one vertex of X and at most one vertex of Y. Since U is 3-connected, there are no such components and U is a $K_{3,3}$. □

The uniqueness of the $(3,3)$-block trees essentially follows. We prove by induction that every $(3,3)$-block tree uses every $(3,3)$-cut in the 2-connected graph. Since when we do our first decomposition every $(3,3)$-cut of G remains a $(3,3)$-cut of one of the pieces in the decomposition. This is straightforward. Furthermore, none of the resulting pieces is $K_{3,3}$ since we've added a triangle on the cut and $K_{3,3}$ contains no triangle. So we can apply our lemma inductively. We also need to show that we get the same set of graph nodes. This is straightforward. For the moment, we omit the details.

We now present three more lemmas of a similar nature:

Lemma 2.3. *Let U be a 3-connected graph that is not $K_{3,3}$. Let X be a 3-cut in U which contains an edge. There does not exist a $(3,3)$-cut separating the vertices of X.*

Proof. Suppose the lemma is false. Let $X = \{a, b, c\}$ where bc is an edge. Let $U_1, \ldots, U_k$ be the connected components of $U - X$. Let Y be a $(3, 3)$-cut separating X. WLOG, Y separates $\{a, b\}$.

Y must contain a vertex of U_1 and a vertex of U_2. Hence, every component of $U - Y$ which contains a neighbour of every vertex of Y must intersect X. But there are three such components and $U[X]$ has only two so this is impossible. □

Lemma 2.4. *Let U be a 3-connected graph that is not $K_{3,3}$. Let X be a $(3, 3)$-cut in U. There does not exist a 3-cut Y separating X such that $G[Y]$ is connected.*

Proof. Suppose the lemma is false. Let $X = \{a, b, c\}$. Let $U_1, \ldots, U_k$ be the connected components of $U - X$. Let Y be a 3-cut separating X such that $G(Y)$ is connected. WLOG, Y separates $\{a, b\}$.

Since $G[Y]$ is connected, Y fails to intersect some component of $G - X$. Thus $X - Y$ is contained in one component of $G - Y$. The other component of $G - Y$ must touch all of Y, and hence $Y - X$ is in one component of $G - X$, a contradiction. □

Although not all of the 2-cuts of a 2-connected graph appear in every decomposition, it turns out that every $(2, 3)$-cut must appear in every decomposition. To prove this by induction, it is enough to show that no $(2, 3)$-cut seperates any other 2-cut. We actually prove something stronger.

Lemma 2.5. *A 2-cut X in a 2-connected graph G whose vertices are joined by 3 internally vertex disjoint paths does not seperate the vertices of any other 2-cut.*

Proof. Suppose $X = \{a, b\}$ seperates $Y = \{y_1, y_2\}$. This means that X and Y are disjoint and hence, because of the three internally disjoint paths between a and b, the vertices of X are in the same component of $G - Y$. Since G is 2-connected, all the other components of $G - Y$ have edges to both vertices of Y. This contradicts our assumption that X seperates Y. □

Suppose now that we have decomposed a 2-connected graph H using all its 2-cuts whose endpoints are joined by three internally disjoint paths. The lemma above tells us that the resultant decomposition tree is unique. Let G be one of the unique pieces into which H was decomposed. Suppose G is not 3-connected and consider any remaining two cut (a, b) within it. Let U_1 and U_2 be the components of $G - a - b$. Now, $G_1 = U_1 + a + b$ and $G_2 = U_2 + a + b$ are connected, so a two cut seperates a and b precisely if it consists of a cutvertex of G_1 seperating a from b and a cut vertex of G_2 seperating a from b. Since $G_1 + ab$ is 2-connected, every cutvertex of G_1 seperates a from b. Thus we can enumerate them as $u_1, .., u_k$ so that u_i seperates $a, u_1, .., u_{i-1}$ from $u_{i+1}, ..., u_k, b$. We note that this set is non-empty because there are no three internally vertex disjoint paths from a to b in G. $G_1 - u_i - u_{i+1}$ has a component containing a, a component containing b, and posibly a third component L_i (which can only be absent if $u_i u_{i+1}$ is an edge). Now, there are two internally disjoint paths of $L_i + u_i + u_{i+1}$ between u_i and u_{i+1} as otherwise there would be a cuvertex of G_1 between u_i and u_{i+1}. So there are three internally disjoint paths of G between these two vertices, and if

L_i existed $\{u_i, u_{i+1}\}$ would form a 2-cut on which we should have already decomposed. This means that L_i does not exist (since we cut any such component off during the decomposition process) and u_iu_{i+1} is an edge. In the same way, au_1 and u_kb must be edges and G_1 is a path. Performing a symmetric analysis of G_2 we see that G is a cycle. Thus H is uniquely decomposable into cycles and 3-connected graphs, and to finish off the decomposition we are free to arbitrarily choose a triangulation of each cycle. We will apply a corollary of this fact later. We note that it implies that the number of vertices of a 2-connected graph which are in two cuts is at most four times the number of two cuts in a decomposition. To see this note that certainly the vertices involved in two cuts whose vertices are joined by three internally disjoint paths is at most twice the number of such cuts, which are all used in the decomposiiton whilst the vertices in a cycle we decompose are at most four times the number of cuts that we use to decompose it. Furthermore, given the 2-block decomposition we can find a set of this size containing all the vertices in 2-cuts. We take all vertices involved in any of our decomposing two cuts, and any vertex of degree two whose neighbours were a 2-cut of the decomposition which split the graph into only two parts. Finding the latter set of vertices given the decomposition tree is easy, and there are at most two per cut used in the decompostion.

3 Cutset structure relative to a reduction

In this section, we study the correspondence between the cuts in a 3-connected graph G and those in the graph H obtained by contracting an induced matching M in G.

To ease notation, we use f to refer to the function from $V(G)$ to $V(H)$ corresponding to the contraction of a fixed matching M. If v is in no edge of the matching then $f(v) = v$, so $H = f(G)$ and $G = f^{-1}(H)$.

Note that if Y is a cut in H then $f^{-1}(Y)$ is a cut in G. Therefore since M is matching and G is 3-connected, H is 2-connected. The converse is almost true for cutsets of size three in G. For any such 3-cut X, $|f(X)| = 3$ or $|f(X)| = 2$. If $f(X)$ is not a cut in H, then for some component U of $G-X$ we must have that $f(U) \subseteq f(X)$. If U has more than one vertex then there is an edge uv of U. This is not an edge of M as otherwise $f(U) \not\subseteq f(X)$. Since M is induced, we can therefore assume WLOG that $f(u) = u$. But, again, we contradict that $f(U) \subseteq f(X)$.

So U is a vertex u. Since $f(U) \subseteq f(X)$, we know that xu is an edge of the matching M for some vertex x of X. Since M is induced it follows that $f(v) = v$ for every other neighbour v of u and hence $N(u) \subseteq X$. Since u has at least three neighbours, it follows that u has exactly three neighbours x, y, z in G and $X = \{x, y, z\}$. Thus the only cutsets of size three in G whose image in H is not a cutset are $N(u)$ for vertices u of degree three which are in the matching and for which $G - u - N(u)$ is connected.

We are actually more interested in which $(3, 3)$-cuts of G do not correspond to $(2, 3)$-cuts or $(3, 3)$-cuts of H. If this is to happen then for some component U of $G-X$, $f(U) \subseteq f(X)$.

Mimicing the arguments above, we see that this occurs precisely if $U = \{u\}$ and $X = N(u)$for some vertex u which has degree three, is in the matching and for which $G-u-N(u)$

has exactly two components. Note that in this case $f(X)$ is a $(3,2)$-cut of H which induces a P_3 or a triangle.

In summary then, the $(3,3)$-cuts of G are of one of the following types:

1. $f(X)$ is a $(2,3)$ cut of H
2. $f(X)$ is a $(3,3)$ cut of H,
3. $f(X)$ is $(3,2)$ cut of H which induces a connected subgraph of H, and $X = N(v)$ for some vertex v of degree 3 in G which is in the matching we contract.

4 Refining Our Decomposition

We assume now that we have constructed a 3-connected graph H, by contracting an induced matching M in a 3-connected graph G with at least 17 vertices. and constructed a $(3,3)$-block tree of H and a planar embedding of each piece of H (note since L contains no triangle, and H is not L, no piece of the decomposition is L). We now attempt to add edges to the graph nodes of our decomposition, maintaining the planarity of its nodes, so that every $(3,3)$-cut of G corresponds either to a cutting node of the tree or to a seperating triangle in one of the graph nodes. If we fail, then we will find and return a K_5 model. Since H is 3-connected, we need only worry about the $(3,3)$ cuts which correspond to $(3,2)$ cuts of H which induce a P_3, such that the midpoint of this edge is xv for a vertex v of degree three in H which is also joined to the endpoints of the P_3 in G. We claim that we can add edges between the endpoints of all such P_3s to turn them into triangles, whilst maintaining the planarity of the graphs corresponding to the graph nodes of our tree decomposition.

To begin, we note that since such P_3s are connected 3-cuts of H, they are not seperated by the $(3,3)$ cuts of H. Thus, each of them is contained in the subgraph G_t corresponding to some graph node. Furthermore, t is unique (since the intersection of any two graph nodes is a clique of size at most 3) and the P_3s are 3-cuts of the G_t as well (since they do not seperate the $(3,3)$-cuts, they cannot disappear as we perform a decomposition). This implies that the vertices of the P_3 do not all lie together on some face of the (unique) embedding of G_t, as otherwise by adding a vertex in this face adjacent to all of the vertices of the P_3 we would obtain a planar drawing of a graph with a $K_{3,3}$ minor. It also implies that there is face containing the endpoints of the P_3. Indeed an (induced) P_3 is a cutset of the 3-connected planar graph G_t precisely if there is a face containing its endpoints but not its midpoint.

We see then that for any specific P_3 corresponding to a $(3,3)$-cut of G, we can add an edge between its endpoints to the graph node which contains it whilst maintaining the planarity of this graph node. We now want to prove that we can add all such edges at the same time. It is enough to show that for every face of the embedding there do not exist two P_3s abc and def with no endpoints in common such that a, d, c, f appear in the given cyclic order around the face of the embedding. This is true if the midpoints are distinct, as we can draw the cycle $adcf$ in the face, along with a new vertex adjacent to all four of a, c, d, and f, which in this case would yield a planar drawing of a graph with a K_5 minor. So, we need only consider two P_3s with the same midpoint. In this case, the midpoint is the image of a matching edge which joins two vertices of degree three each of whose neighbourhoods is a $(3,3)$ cut. In this case, it is easy to verify that the corresponding $(3,3)$ cuts of G cross which is impossible. Thus we can indeed add all the edges to make these P_3s into triangles. We now show how to do so in linear time.

We need only generate a list of edges which we need to add, since for any graph node t and corresponding subgraph G_t with the extra edges added, finding a planar embedding can be done in linear time and since G_t is 3-connected it remains so when we add these edges and this embedding will be unique.

For each node v of degree three in G which is in the matching there is a unique P_3 of H corresponding to the unique P_3 of G with midpoint v which does not contain the matching edge incident to v. This is the P_3 which will correspond to $N(v)$ if it is a $(3,3)$ cut of type 3. Furthermore, $N(v)$ is a $(3,3)$ cut precisely if the image of this P_3 is a $(3,2)$ cut of H. This happens precisely if for the unique t such that G_t contains the P_3, its endpoints lie together on a face not containing its midpoint. Thus, our problem reduces to determining for each G_t which of at most $|V(G_t)|$ P_3s of G satisfy this property. We describe how to do so below.

We now show how to determine, in linear time, for which of a list of $O(|V(F)|)$ P_3s in a planar 3-connected graph F, there is a face containing the endpoints of the P_3 but not the midpoint. In a 3-connected planar graph G, every two nonadjacent vertices lie together in at most one face. We will seperately check for which P_3s some face contains their endpoints and for which P_3s there is a face containing all three vertices of the P_3. Because G is 3-connected the latter occurs precisely if the P_3 is part of the boundary of this face in the unique embedding of G_t. But we can check this by generating an embedding and enumerating all the P_3s on its face boundaries (note we sort these lists lexicographically before comparing them, to do so we use bucket sort, which is linear in the size of the graph). So the second problem can be solved in linear time, it remains to solve the first.

We are actually asking which of the endpoint pairs have a common neighbour in the face-vertex incidence graph. We can generate the face vertex incidence graph in linear time, and it has linear size. Furthermore, it is planar and so has degeneracy at most 5. We show now that for any graph J of degeneracy bounded by some constant C, and list of $O(|V(J)|)$ pairs of vertices we can determine in $O(|V(J)|)$ time, all of the common neighbours of all these pairs. We consider an ordering $<$ of V for which each vertex has at most C neighbours appearing before it in the order. For every w, there are at most C^2 pairs of vertices both of which appear before w in the order and have w as a common neighbour. So we can determine, for each pair on the list, all their common neighbours appearing after both of them in linear time. It remains to determine for every pair u, v with $u < v$, the set of common neighbours which are less than v. There are at most C neighbours of v appearing before v, which makes this task easy. Formally, as we strip vertices of degree at most C out of G to generate our order, we can orient the edges of G so that each edge xy with $x < y$ is directed from x to y. Then, we can traverse the edge list of G once to obtain for each v the list $N^-(v)$ of vertices $w < v$ such that wv is an edge of G. Now we can traverse our set of vertex pairs, replacing a pair (u, v) with $u < v$ with the at most C pairs uw for each w

in $N^-(v)$. Next we can traverse the edge list of G again to see which of these new pairs are edges of G. Finally u and v have w as a common neighbour which appears before w if and only if we found that uw is an edge and that $w \in N^-(v)$. This completes the description of the linear time algorithm (again, before comparing the lists we sort them using bucketsort which takes linear time).

Our next step will be to modify the decompostion tree to take these $(3, 3)$-cuts into account. Basically when we uncontract the midpoint ux of the P_3 and do the cut on $N(u)$, we turn the P_3 into a triangle (which we have already done), and then replace the vertex of H corresponding to the matching edge by x. This latter step does not change H at all because after turning the P_3 into a triangle this vertex has the same neighbourhood in G and H as the old contracted vertex did. What we do need to do is to split apart the graph into pieces using all the cuts corresponding to these seperating triangles in our extended planar embedding which have now become $(3, 3)$ cuts. We note that some of these correspond to P_3s we have turned into triangles and others correspond to triangles in the embedding of G_t.

In doing so, we exploit the fact that each G_t has a unique planar embedding. Thus, each non-separating triangle is a face. So, since we have a planar embedding of the graph, to generate all the separating triangles we need only generate all the triangles. But, for an edge uv, we have that uvw is a triangle precisely if w is a common neighbour of u and v. We use the algorithm described above to generate the set of such w for every edge uv, thereby obtaining the set of triangles of G.

We then check which of these separating triangles corresponds to a $(3, 3)$ cut X of G which is $N(v)$ for some vertex v of degree three which is in the matching. This simply involves looking at the preimage of each separating triangle and can easily be done in linear time. We obtain a set of separating triangles on which we will split G_t. Adding a cutting node and a graph node with vertex set $v + N(v)$ incident to it for each such triangle.

We now have to partition the graph further, because each separating triangle which becomes a 3-cut will cause us to split our planar embedding into two smaller embeddings, with our separating triangle as a face in each of of these embeddings. A linear time algorithm to do so has been developed by Havet, Quercini, and Reed[3]. It uses a breadth first search of the face vertex graph of a planar embedding, exploiting the fact that the vertices of a separating triangles all lie on the same level of this tree or on levels i and $i + 2$ for some i. They treat each such pair of consecutive levels separately to solve the problem (This can be done in two ways, either using the fact that the graph under consideration has bounded tree width or via a more combinatorial approach).

5 Component graphs

In this section, we describe a linear time algorithm which will construct the $(3, 3)$-block tree for G given a $(3, 3)$-block tree for a 3-connected graph H obtained from G by contracting the edges of an induced matching all of whose vertices have degree between 4 and d. Further, we suppose that every $(3, 3)$-cut of G can be obtained from taking a subset of 3 vertices of $f^{-1}(Y)$ for some $(3, 3)$-cut Y of H.

Suppose Y is a $(3,3)$-cut in H and we want to determine if a subset S of three vertices of $f^{-1}(X)$ is a $(3,3)$-cut in G. If the vertices in $X - S$ connect the components of $G - X$ into fewer than three components then S is not a $(3,3)$-cut in G. Otherwise, S is a cut in G.

This concept is formalized using the notion of a component graph.

Definition 5.1. Let G be a graph and Z a subset of the vertices of G. Let $C_G(S)$ be the (simple) graph obtained from G by contracting all the edges with no endpoint in S. $C_G(Z)$ is called the *component graph* of Z in G.

A vertex in $C_G(Z)$ that is not in Z is called a *component vertex*.

$S \subseteq Z$ is a $(3,3)$-cut in G if and only if S is a $(3,3)$-cut in $C_G(Z)$. However, $C_G(Z)$ is typically much smaller then G.

We outline the main algorithm of this section.

For each cutting node t of the $(3,3)$-block tree of H,

1. Build $C_H(H_t)$.
2. Build $C_G(f^{-1}(H_t))$.
3. Determine which sets of three vertices of G contained in $f^{-1}(H_T)$ are $(3,3)$ cuts of G by examining this graph.
4. Construct the $(3,3)$-block tree for G by combining the information that we have obtained.

Building the component graph for H_t is easy, since the components of $H - H_t$ correspond to the components of the decomposition tree when we remove t and edges out of H_t to these components have already been assigned to the nodes of the tree. We note further that the vertices in the cut have bounded degree so that the total size of this component graph is constant. Further, if we are given the component graph of H_t in H, we can easily obtain the component graph of $f^{-1}(H_t)$ in G. To do so, we simply uncontract the vertices contained in $f(M) \cap H_t$ in $C_H(H_t)$. We then look at the neighbourhood of edges of the vertices in $f^{-1}(H_t)$ to decide their endpoints in $C_G(f^{-1}(H_t))$. This can be done with a list of the vertices in G with pointers to their respective component vertex in the decomposition for H.

To build the $(3,3)$-block tree of G, we root the $(3,3)$-block tree of H. We then use dynamic programming, examining this tree from the leaves to the root, to build the $(3,3)$-block tree of H. We first describe a quadratic time algorithm to do this and then improve it ot a linear time algorithm.

We traverse the $(3,3)$-block tree of H in a post-order traversal. When we examine a cut node, uncontract everything (of the matching) in this cut node or below it that is not already uncontracted. For every child graph node cut off from the parent graph node by a $(3,3)$-cut contained in the 3 cut of this cutting node, make a graph node of the new $(3,3)$-block tree. Merge the remaining graph nodes into the parent graph node.

This algorithm may take quadratic time as we may merge the same set of vertices multiple times. To make this a linear time algorithm, we replace these child graph nodes with an auxiiary reminder vertex encoding what the merge we should do. When we complete our traversal of the $(3,3)$-block tree, we consider the tree in postorder repeatedly expanding a graph node by replacing the auxiliary vertices corresponding to children of a particular cut node by the corresponding graph nodes.

6 Massaging the Matching

In this section, we suppose that for some $d > 500$ and $\varepsilon = d^{-6}$, we are given a matching M of size $d^4\varepsilon|V|$ in G, all of whose vertices have degree at most d in G. We construct from this matching either an induced matching N with at least εn edges such that contracting the edges of N in G yields a 3-connected graph, or a set of 3-cuts of G each of which contains an edge and none of which seperates another, which decompose G into at least εn pieces.

We outline our approach and provide details later in the section. We start by finding a large enough submatching M' of M such that that no vertex of degree at most six has a neighbour in two distinct matching edges. We will work with M' rather than M. We contract M' to obtain H and build its 2-block tree $[T.\mathcal{H}]$. H is 2-connected since G is 3-connected and M' is a matching. We find our desired structure (a new matching N or a set of 3-cuts) within nodes of the 2-block tree that contain $< \varepsilon^{-1}$ vertices of H and ignore the remaining nodes. How we find our structure depends on the properties of H and T.

- If H does not have many vertices contained in a 2-cut then we obtain a submatching of M' by removing edges whose contraction creates a 2-cut from M'. This submatching is large enough and can be contracted to obtain a 3-connected graph.
- If T has many leaves, then we consider the (nearly disjoint) subgraphs of G obtained by uncontracting the matching edges in each leaf graph node of the decomposition. For each leaf we will find either a 3-cut which is contained wholly within this subgraph, or an edge in this subgraph which is not contained in a 3-cut of G and whose deletion leaves the subgraph connected. We either find many 3-cuts and return them, or find many edges which we use to form a matching.
- If T has few leaves but many vertices in 2-cuts then the 2-block tree has few internal nodes of degree > 2 and therefore many nodes of degree 2. We use these degree 2 nodes to find many long node-disjoint paths in T. As in the previous case, we then choose for each path, either a 3-cut within the path or an edge of the path which is not in any bothersome cutsets.

We now provide the necessary details.

Since the maximum degree of the vertices in the matching is d, we can greedily find a submatching of size at least $\frac{|M|}{12d} > 30d^2\varepsilon n$ which is induced and such that no vertex of degree at most six has a neighbour in two distinct matching edges. We contract down the edges of this matching to form a graph H. We then form the 2-block tree of H. If there are fewer than $12d^2\varepsilon n$ 2-cuts used in the decomposition then the set of vertices of H contained in a

2-cut has size less than $24d^2\varepsilon n$. In fact, these vertices are contained in the set of all vertices contained in a cutting node of a 2-block tree as well as any vertex contained in a graph node that is K_3 such that the other 2 vertices are contained in a cutting node of degree 2. In this case, we find all the vertices in $f(M)$ in this set and remove the corresponding matching edge from M. We are left with a submatching of size greater than εn whose contraction yields a 3-connected graph. So, we can assume that there are at least $12d^2\varepsilon n$ 2-cuts used in the 2-block tree.

Consider now a leaf of the 2-block decomposition of H which has at most ε^{-1} vertices and the corresponding subgraph L of G obtained by uncontracting the edges of M in it. For each such leaf, we will chose a 3-cut of L which contains an edge or an edge of L which is contained in no 3-cut of L and the deletion of its endpoints does not disconnect L. Our choice depending on whether the 2-cut Y of H cutting this leaf off uncontracts to a set of three vertices or a set of four vertices.

If $f^{-1}(Y)$ has only three vertices then we simply note that $f^{-1}(Y)$ is a 3-cut containing an edge seperating L from a component containing the rest of G. We choose this 3-cut for this leaf.

If $f^{-1}(Y)$ has four vertices then, since G is 3-connected and no vertex of degree at most five in G has neighbours in two different matching edges we see that L has at least 7 vertices. We try to find two paths of L joining the two edges of $f^{-1}(Y)$. If we fail, we find a 1-cut seperating them. This one cut forms a 3-cut with one of the edges in $f^{-1}(Y)$ seperating off some of $L - f^{-1}(Y)$ from the rest of G. If we find two paths then we can also find an edge of $L - f^{-1}(Y)$ for which the deletion of its enpoints leaves a path between the two edges of $f^{-1}(Y)$ (as we can use an edge on one path unless they are both of length three in which case our degree condition says that each midpoint is joined to some other vertex in L). We check if this edge is contained in some 3-cut of L. If so, we choose this 3-cut which is also a cut of G. Otherwise, we choose this edge.

If the block tree for H has at least $4\varepsilon n$ leaves then it has at least $2\varepsilon n$ leaves such that L has at most ε^{-1} vertices (since the graphs corresponding to these leaves are disjoint, except for the cut Y). For each of these, we chose a 3-cut or an edge as above. We can find these in constant time per leaf since the graphs under consideration have size $O(1)$. We note that the 3-cuts we choose do not seperate each other (for this to happen, for some leaf L cut off by Y, a cut not corresponding to Y would have to choose a vertex in $L - f^{-1}(Y)$ and this never occurs). So if there are εn leaves for which we chose a 3-cut, we return this family. If we chose εn edges, we note that no two of these edges form a cutset of G because removing an edge leaves the rest of the leaf connected, so there is only one component touching it and it cannot be in a minimal cut which is otherwise disjoint from the leaf. Furthermore, these edges form an induced matching as the corresponding vertices of H are not adjacent. In this case we return this set of edges.

So we can assume that there are at most $4\varepsilon n$ leaves in the 2-block tree of H. It follows that there are at least $11d^2\varepsilon$ nodes of degree 2 in the decomposition tree and we can partition these into a family of $8\varepsilon n$ paths. We delete any graph node which has more than ε^{-1} vertices from this partition. We still have a partition of at least $10d^2\varepsilon n$ nodes of degree 2 into at most

$10\varepsilon n$ paths. We find within these a set of εn disjoint paths each of length at least 10000.

We will use these paths to find the desired induced matching. It suffices to find in each of these paths an edge of G which does not appear in the block tree outside of the path and such that deleting it leaves the graph corresponding to the path connected. these edges form the desired induced matching.

We partition each path up into 5 disjoint subpaths each of length at least 1000 which begin and end with a cutting node. For each subpath we consider the subgraph of G which is its preimage. We note that this graph contains at least 5000 vertices of H in its cutting nodes since at each cutting node, we need to change the cut, and no vertex of H which leaves the cut can return to it. Since H is two connected, it has two paths from the vertices in the first cutting node to those in the last cutting node. One of these paths must have at least 500 nodes, and corresponds to a path of G which is at least as long. These paths show that the subgraph of G is either connected or has exactly two components. If all five of the subpaths correspond to connected graphs then we will consider the middle edge from the longer path in the subgraph corresponding to the middle path. This edge does not contain any of the eight vertices in the preimage of the cuts at the two ends of the path. Thus it does not appear outside the path. We check if there is a 3-cut of G consisting of this edge and another vertex in the (original long) path. We can do so easily since we know that such a cutset must be a cutset separating some of the middle path vertices from the rest of G or separating the four cuts corresponding to the two ends of the paths (hence we consider only a finite sized graph). If there is no such 3-cut then we use the edge under consideration in the matching. Since the other subpaths also correspond to connected graphs, it is easy to verify that no such edge can form a 3-cut or a four cut with an edge outside its path.

Suppose now that one of the subpaths has two components. Then we restrict our attention to the component containing the longer of the two paths of H connecting the cutting nodes at the endpoints. Thus, we have a path of H which contains at least 500 cutnodes such that the only other attachments to H are from its endpoints. Since G is 3-connected, all but at most two of these cutvetices correspond to edges of H, and more strongly, there are at most two cutvertices of G in this graph. Thus we can find a subpath of our path containing 15 vertices of H which correspond to matching edges which are cutvertices of this subgraph of H and whose endpoints are linked by two disjoint paths of G each of length at least 15. We delete the middle edge of one of these paths. We can now mimic the argument from the previous case, the three components will be attached before we leave this part of the graph because of all the matching edges touching both paths. Thus we either find a 3-cut using this edge in this small part of the graph or we can put this edge into the matching.

7 Decomposing on 3-cuts

Suppose the algorithm in section 6 returns a family of 3-cuts each of which contains an edge and none of which separates each other. In this section, we explain how these 3-cuts are used to decompose G into smaller graphs and how to paste together $(3,3)$-block trees for the samller graphs to form a $(3,3)$-block tree of G. Contracting the edges in each of the

3-cuts turns these 3-cuts into 3-cuts in the resulting graph. We build the 2-block tree of this graph, obtain the corresponding components graphs and build the components graphs for the original 3-cuts as described in section 5. We can build a special 3-block tree of G which uses only these 3-cuts. This can be done in the same way we build the $(3,3)$-block tree described in section 5.

The procedure of section 6 actually returns a set of 3-cuts each containing an edge such that the edges under consideration form a matching and no two of these edges cross. We contract the matching and add an edge between the vertices of each 2-cut corresponding to a 3-cut we are interested in. We build the 2-block tree for this contracted graph and then coarsen it by removing any cuts except the ones we are interested in. This splits our problem into subproblems. We actually put a triangle on each of these cuts when reducing to the subproblems (since there is an edge in the cut this does not affect the existence of a K_5 minor). We now have to combine the solutions (assuming these are $(3,3)$-block trees) to obtain a solution for the whole graph.

Note that one of our cuts is a $(3,3)$-cut of G precisely if it is a $(2,3)$-cut after the contractions. If this is the case, we simply paste our solutions together using a cutting node. If the 3-cut is not a $(3,3)$-cut then we will paste two planar embedings together on triangular faces to get an embedding of the union of the two graphs with a separating triangle. We then delete all the edges of triangles used in such pastings which are not in G.

8 Algorithms

In this section, we elaborate on the steps of the algorithm which builds a $(3,3)$-block tree of a 3-connected graph G.

8.1 Removing small degree vertices

We describe a linear time algorithm which determines if such a set exists and if so finds one.

Algorithm 8.1.

- Arbitrarily label the vertices of G from 1 to n. For each vertex v of degree 3, build a triple $t_v = (i_1, i_2, i_3)$ corresponding to the labels of vertices in $N(v)$. Sort each triplet t_v so that $i_1 < i_2 < i_3$.
- Bucket sort the t_v's on each coordinate in turn while keeping the ordering from our previous sort.
- Let L be the sorted list of triplets. Note that neighbourhoods that are equal (same value for all coordinates) now occupy a contiguous region in the list.
- Remove all but 4 copies of every triplet in L that appears more than 3 times and remove their corresponding vertices in G. Call the resulting graph H.

We note that the neighbours of the vertices which we remove all have degree at least 4 in G so there is no danger that they are also removed. Indeed they have degree at least 4 in H, so since G has minimum degree three (it is 3-connected) so does H, indeed H is also 3-connected and every $(3,3)$-cut of G is also a $(3,3)$-cut of H. We also remark that when adding the new graph nodes corresonding to the deleted vertices of G to the decomposition tree for G we will bucket sort the triples corresponding to the cutting node for the tree, and then add the desired edges by comparing these two sorted lists.

8.2 Adding small degree vertices to the decomposition

In this section, we describe a linear time algorithm which takes the $(3,3)$-block tree of the graph H and the list L of degree 3 vertices, both obtained by the previous algorithm and returns a $(3,3)$-block tree of G.

Algorithm 8.2.

- Keep the triplet and labelling of the vertices of G from algorithm 8.1. For each cutting node of the $(3,3)$-block tree of H, build a triplets (i_1, i_2, i_3) of its vertices with $i_1 < i_2 < i_3$. Call this new list L'.
- Merge the two lists of triplets, but mark the elements of L' before doing so.
- Bucket sort the merged list on each coordinate in turn while keeping the ordering from our previous sort.
- Now the triplets that have the same value for all coordinates occupy a contiguous region in the sorted list. Note that each region of equal triples contains exactly one marked triplet (corresponding to a cutting node).

  Passing through the list once, put the marked triplet at the begining of the region of equal triplet.

  For each unmarked triplet, build a graph node of the corresponding vertex with its neighbourhood. Add an edge between this graph node and the cutting node corresponding to the triplet of the same value (if we read through the list, this is the last marked triplet).

Return the $(3,3)$-block tree with these added nodes.

8.3 Finding a K_5-model

In this section, we describe a linear time algorithm which finds a K_5-model in a non-planar 3-connected graph G with no $(3,3)$-cut with more than 8 vertices.

We run the algorithm from [10] to obtain a $K_{3,3}$ or K_5 subdivision. If we obtain a subdivision of K_5, we return it as a K_5-model. So suppose we obtain a $K_{3,3}$ subdivision S of G. Let $a_1, a_2, a_3, b_1, b_2, b_3$ be the centres of this subdivision labelled so that $A = \{a_1, a_2, a_3\}$ correspond to vertices on one side of the bipartition and $B = \{b_1, b_2, b_3\}$ correspond to the vertices on the other. We let $Q_{i,j}$ be the interior of the path of H between a_i and b_j.

We will now use the $K_{3,3}$ subdivision to find either a subdivision of L or a K_5-model. We will then obtain a K_5-model from the L-model in the first case.

We first describe the algorithm which finds either a subdivision of L or a K_5-model given a subdivision H of $K_{3,3}$.

Algorithm 8.3. We note that the graph obtained from $K_{3,3}$ by subdividing two disjoint edges and adding an edge between the two new vertices is L.

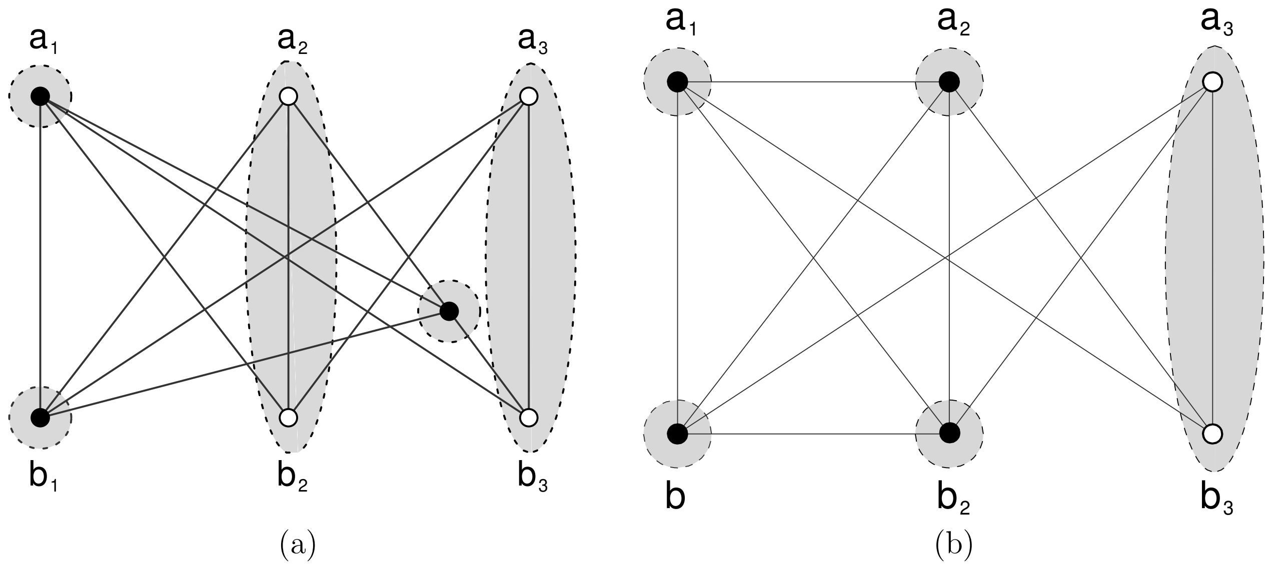


Figure 5:

Since G is 3-connected, $G - A$ has at most 2 components. So at least two vertices of B are in the same component. Build $G - A$ and DFS starting from each b_i to find a path P between two vertices of B. Relabel the endpoints of P by b_1 and b_2. Build $H - A$ and find a path P_1 between the component containing b_1 and the component containing b_2. This can be done by walking along P (in $G - A$) starting at b_1 until we reach a vertex in the component containing b_2 while remembering the last vertex in the component of b_1 that we visited. P_1 is internally disjoint from H.

If P_1 is vertex-disjoint from $H - B$ then symmetrically find a path P_2 between components (containing different vertices of A) of $H - B$. If P_2 is vertex-disjoint from $H - A$ and P_1 does not intersect P_2 then we have found a K_5-model by contracting the (subdivided) edge between b_3 and the vertex of A not in P_2 as in 5(b). If P_2 is vertex-disjoint from $H - A$ and P_1 intersect P_2 then we build a K_5-model where we use H as a K_4-model centered at the endpoints of P_1 and P_2 and the last vertex image consists of a spanning tree for the connected graph formed by the union of the interiors of P_1 and P_2. If P_2 is not vertex-disjoint from $H - A$ then we swap the labels of (A, P_1) and (B, P_2) so that P_1 is not vertex-disjoint from $H - B$.

Build $(H \cup P_1) - B$ and count number of components in it. If this graph has only two components then P_1 links the interior of two paths of H corresponding to disjoint edges of the $K_{3,3}$ since P_1 links two components containing different vertices of A in $H - B$. Thus $H \cup P_1$ is a subdivision of L. Return it.

Now suppose $(H \cup P_1) - B$ contains 3 components. The component containing P_1 also contains a vertex of A. Relabel A so this vertex is a_1.

By our choice of P_1, we can assume one of the endpoints of P_1 is on $Q_{1,1} - a_1$ and the other is on $Q_{1,2} - a_1$. Use DFS to find a path in G joining 2 components of $(H \cup P_1) - B$. Such a path exists since otherwise B would be a $(3, 3)$-cut in G. As before, we choose a subpath P_3 of this path which is disjoint from $(H \cup P_1) - B$.

If P_3 has one endpoint on the interior of P_1 then there is a path P in $P_1 \cup P_3$ linking the interiors of two paths of H corresponding to disjoint edges of the $K_{3,3}$. Then $H \cup P$ is a subdivision of L. Return it.

If P_3 has joins two vertex disjoint edges then $P_3 \cup H$ contains a subdivision of L. Return it.

Suppose next that the interior of some path $Q_{i,j}$ contains both an endpoint of P_1 and an endpoint of P_2. Then $H \cup P_1 \cup P_2$ contains a minor of the graph of Fig. 5 (a), and hence a K_5 minor. Return the K_5-model corresponding to this minor.

If neither of the last three possibilities occur then $H \cup P_1 \cup P_2$ contains a minor of the graph of Fig. 5(b) and hence a K_5 minor. Return the K_5-model corresponding to this minor.

We now describe the algorithm which given an L-model of G, finds a K_5-model.

Algorithm 8.4. Note that L is simply a C_8 with all long diagonals added. We claim that adding any edge e between non-adjacent vertices of L yields a graph with a K_5 minor. By symmetry, we are free to choose the first endpoint of e to be v_1. Now we need only consider $e = v_1v_3$ or $e = v_1v_4$, as $e = v_1v_7$ is symmetric to the first case and $e = v_1v_6$ is symmetric to the second. In the first case there is a K_5 model with vertex images $v_1, v_3, v_2v_6, v_4v_5, v_7v_8$ (see Fig 6(a)). In the second there is a K_5 model with vertex images $v_1, v_4, v_2v_3, v_5v_6, v_7v_8$ (see Fig. 6(b)). Thus our claim is proved.

We now claim subdividing an edge uv of L and adding a new edge between the new vertices x and another vertex y of L yields a graph which contains a K_5 model. One of u or v is not adjacent to y since L contains no triangles. Contract the edge between that vertex and x. This yield L with an added edge (between y and u or v). Note also that adding an edge between two subdivided edges of L also yields a graph with a K_5-model as we can contract any half of a subdivided edge to reduce it to the previous case.

Let H be the L-model given in the input.

Since G contains more than 8 vertices, either one of the edges in the model contains a vertex or there is a vertex in $H - G$.

In the first case, suppose the subdivided edge is Q with endpoints uv. Use DFS to find a path P in $G - u - v$ from Q to $H - Q$. Such a path exists as G is 3-connected. Apply the above remark to $H \cup P$ to find a K_5-model and return it.

In the second case, use DFS to find a path P_1 in G from some vertex v in $H - G$ to some vertex in H, say h_1. Then use DFS to find a path P_2 in $G - h_1$ from v to some vertex h_2 in $H - h_1$. $P_1 \cup P_2$ is a walk from h_1 to h_2 that is internally disjoint from H (we stop our DFS once we reach H). Take a subwalk P of $P_1 \cup P_2$ that is a path. If h_1h_2 is an edge, replace it by P and apply the algorithm described in the previous paragraph. If h_1h_2 is not an edge

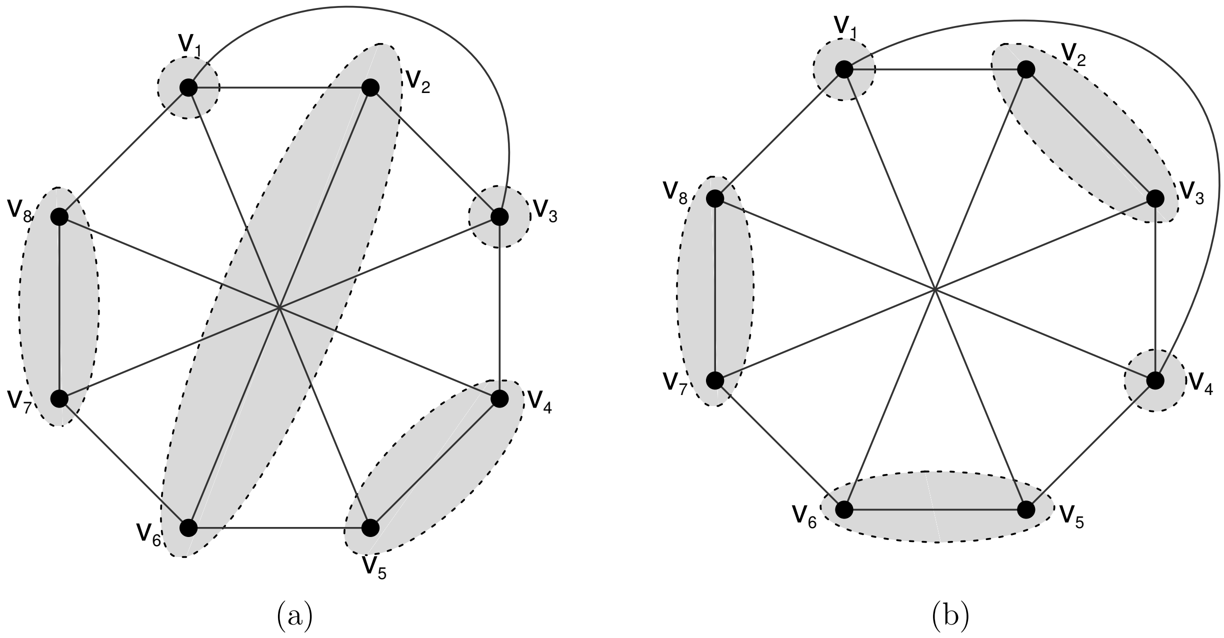


Figure 6:

then $P \cup H$ is a model of L with an added edge. We find a K_5-model as described in the first paragraph and return it.

8.4 Finding a matching

Let G be a 3-connected graph. Let S_i be the set of vertices in G of degree i. Let $S = \bigcup_{i=3}^{d} S_i$.

We begin by describing an algorithm to find a maximal matching M in the graph induced by S.

Algorithm 8.5. Let M be the empty graph.
  For $v \in V(S)$, let i be unmarked.
  For each edge $uv \in E(S)$
    If u is unmarked and v is unmarked then
      Add u and v to $V(M)$
      Add uv to $E(M)$
      Mark u and v.

We claim that M is maximal in S. Suppose not. Let $uv \in E(S)$, $u, v \notin M$. When uv is considered in the algorithm, at least one of u or v was marked. So one of u or v was in M. But no vertex added to M is later removed. Contradiction.

We now give an algorithmic proof of the following statement from section 1.3.

Theorem 8.6. *Let G be a graph of minimum degree three with at most $64|V(G)|$ edges. Let $d = 100000$ and let epsilon $= d^{-6}$. In linear time we can find one of the following:*

1. *A set of at least $5\varepsilon|V(G)|$ vertices of degree 3 which have the same neighbourhood as at least $d+1$ other such vertices.*

2. *A matching M in the subgraph of G induced by the vertices of degree at most d which has size $\geq d^4\varepsilon|V(G)|$.*

3. *A minor G' of G such that $|E(G')| \geq 64|V(G')|$, or G' is a subdivision of K_5, in which case we find and return a K_5 model in G.*

Proof. We proceed as follows. We delete a maximum matching from S. It has fewer than $d^4\varepsilon|V|$ edges as otherwise we are done. Now, the vertices of this matching along with the vertices of degree exceeding d form a hitting set B for the edges of G. Since G has at most $64|V(G)|$ edges, it follows that B has at most $\frac{|V(G)|}{d}$ vertices. Every vertex of $V-B$ has at least three neighbours in B. For each of these vertices in turn, we try and choose a pair of their neighbours which has not yet been chosen for a vertex already considered. We can check this condition easily because the vertex has bounded degree. If we choose $\frac{64|V|}{d}$ such edges then they correspond to the subdivided edges of a subdivision of a high density graph and we return this high density minor. Otherwise we construct a subdivision of a graph F with fewer than $\frac{64|V|}{d}$ edges, using some subset S^* of at most $\frac{64|V|}{d}$ vertices of S. We note that the neighbourhood of every vertex of $S-S^*$ forms a clique in F. Thus, if any such vertex has degree exceeding three then we have found a K_5 minor. So, every vertex of $S-S^*$ sees a triangle of F.

We consider F, more closely. We either find a subgraph of F of minimum degree 64 or construct an order in which each vertex has at most 64 neighbours appearing earlier in the order. If we find a high density subdivision again we return it. If we find the ordering we use it to note that F contains at most $\binom{64}{2}|V-S|$ triangles. Thus there are at most $4\binom{64}{2}|V-S|$ vertices in S' whose neighbourhbood is not shared with at least four other vertices of S'. So, we can find a set of greater than $\frac{|V|}{2}$ vertices of degree three which share their neighbourhood with at least five other vertices. We construct an ordering on the vertices of F by repeatedly ripping out a minimum degree vertex. If we never rip out a vertex of degree exceeding 64 we have the desired ordering. Otherwise we have the desired high density subgraph. □

8.5 Putting it all together

We now prove our main result

Theorem 8.7. *There exists an $O(m+n)$ algorithm which, given a $(3,3)$-connected graph G, either*

- *Returns a K_5 of G if G has a K_5 minor or*
- *Returns a $(3,3)$-block tree of G if G has no K_5 minor.*

Proof. We will describe a recursive algorithm.

Use a brute force algorithm for the base case.

Compute the degree of all vertices in G.

Use Fact 1.8 to check if G is 3-connected. If not, use Fact 1.8 to find a block tree of G and 2-block trees for each graph node of the block tree. Recursively run this algorithm on each (3-connected) graph node of the 2-block trees. If a K_5-model was returned by for any graph node, return that model, otherwise, combine the block trees into a CBD for G. Return this CBD.

From now on, we will assume that G is 3-connected.

If $|E(G)| > 64|V(G)|$ then by fact 1.9, we can construct a K_5-model of G in linear time. Return that model.

Now assume that $|E(G)| \leq 64|V(G)|$. We actually insist that this condition also holds for all the graphs we reduce to, as we can easily check if it does before reducing and if not find our desired minor.

Let S be the set of vertices of G with degree $< d$, ≥ 3.

Run the algorithm of Theorem 8.6. It returns either a K_5-model, a set of vertices of degree three, or a matching.

If it returns a K_5-model, return the uncontraction of that model.

If it returns a set of vertices of degree three, run algorithm 8.1 to construct a graph G' and recurse on that graph. If we return with a K_5-model, return that model. If we return with a $(3, 3)$-block tree, add the vertices of $G - G'$ to the block tree as described in section 8.2.

If it returns a matching, using the techniques of section 6, either find a matching M' which is induced and whose contraction leaves a 3-connected graphs or a family F or a set of 3-cuts of G splitting G into at least εn pieces with the properties described in that section.

If M' is returned then contract the edges of M' and call the resulting graph G'. Recursively run the algorithm on G'. If a K_5 model is returned, return the uncontraction of this K_5 model. If a $(3, 3)$-block tree of G' is returned, apply the techniques we have presented to find a $(3, 3)$-block tree of G.

If a family F of cuts is returned then split the graph up using these cuts as described in section 7 and recursively solve the subproblems. If any subproblem returns a K_5-model then this extends to a K_5-model in G. Otherwise we combine their solutions into a $(3, 3)$-block tree of G using the method described insection 7.

If we did not obtain a K_5-model we obtained a $(3, 3)$-block tree for G.

For each graph node Z of the $(3, 3)$-block tree.

Check if it is L (in constant time)

If not, check if it is planar (in linear time [5]).

If not, by fact 1.2, Z contains a K_5-model and, as described in section 8.3, it can be found in linear time. Find this model and return it.

Now we know that G has no K_5 minor. Return the $(3, 3)$-block tree of G.

This completes the description of the algorithm. We turn now to the running time.

8.6 Running time

Our analysis has shown that each iteration runs in $O(c|V|)$ time for some large c (once we preprocess to check if the graph has $O(|V|)$ edges). We claim that this implies that the whole algorithm runs in $O(C(|V|-4))$ time on graphs of size at least five where $C = c\varepsilon^{-1}$. If we recurse by contracting a matching or deleting vertices of degree three, the new subproblem has size at most $n(1-\varepsilon)$ and the bound follows easily by induction. Suppose then, that we break into subproblems using 3-cuts. Any subproblem of size 4 is solved already as the graph is a K_4 so we need spend no time on them. Every subproblem has size at least 4, so if we there are k subproblems of total size N then the total time spent is $CN - C4k + cn$. Now the subproblems are disjoint except on the 3-cut so $n - 3 = N - 3k$ and $k > \varepsilon n$ so $Ck > cn$ and we are again done by induction. We have that the time spend is inductively at most Cn. □

References

[1] Donald S. Fussell, Vijaya Ramachandran, and Ramakrishna Thurimella. Finding triconnected components by local replacements. In *Automata, Languages and Programming*, pages 379–393, 1989.

[2] Carsten Gutwenger and Petra Mutzel. A linear time implementation of spqr-trees. In Joe Marks, editor, *Graph Drawing, Colonial Williamsburg, 2000*, pages 77–90. Springer, 2001.

[3] F. Havet, G. Quercini, and B. Reed. Personal communication, 2007.

[4] J. E. Hopcroft and R.E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 3:135–158, 1973.

[5] John Hopcroft and Robert Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[6] Neil Robertson and P. D. Seymour. Graph minors. XIII: the disjoint paths problem. *J. Comb. Theory Ser. B*, 63(1):65–110, 1995.

[7] Tsan sheng Hsu and Vijaya Ramachandran. A linear time algorithm for triconnectivity augmentation (extended abstract). In *IEEE Symposium on Foundations of Computer Science*, pages 548–559, 1991.

[8] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[9] W.T. Tutte. *Graph theory*. Addison-Wesley Pub. Co, 1984.

[10] S. G. Williamson. Depth-first search and Kuratowski subgraphs. *J. ACM*, 31(4):681–693, 1984.