We update Algorithm 2 and Algorithm 3 of the original working note on the topic, which was written by Shant Karakashian.

1 Related Work

The Karypis Lab has developed a large library\(^1\) of graph partitioning algorithms (MeETIS, ParMETIS, hMETIS). This library is particularly well suited for large graphs. It is commonly used by the UAI community for tree decomposition, and should be checked before doing any implementation work.

\(^1\)http://glaros.dtc.umn.edu/gkhome/views/metis

\[1\] presents a linear time algorithm for minimal elimination ordering approximation in planar graphs.

2 \(\mathcal{O}(n^4)\) Algorithm

Below we store in \(fcount[x]\) the number of fill edges that need to be added to the graph when the vertex \(x\) is removed.

Complexity Analysis

The complexity of Algorithm 1 depends on the complexity of

\[\text{Line 1 in Algorithm 1 (FILLCOUNT). The complexity of FILLCOUNT is determined by the three nested loops: } \mathcal{O}(n^3).\]
Algorithm 1: MinFill(G)

**Input:** A graph $G = (V, E)$, where $|V| = n$.
**Output:** Perfect elimination order $\sigma$

1. **FILLCOUNT(G)**
2. for $i = 1$ to $n$ do
3.     $v \leftarrow$ the vertex in $G$ with the smallest value of $fcount$
4.     $\sigma[i] \leftarrow v$
5.     AddFillEdgesAndRemoveNode($G, v$)
6. return $\sigma$

Algorithm 2: FILLCOUNT(G)

**Input:** A graph $G = (V, E)$.
**Output:** Vertices labeled with the fill count, which is the number of edges that need to be added to make the vertex simplicial

1. foreach $v \in V$ do
2.     $neigh[] \leftarrow$ Neighbors($v$) /* Array storing neighbors of $v$ */
3.     $count \leftarrow 0$
4.     for $i \leftarrow 1$ to Size($neigh[]$) do
5.         foreach $j \leftarrow i + 1$ to Size($neigh[]$) do
6.             if ($neigh[i], neigh[j]$) $\notin E$ then
7.                 $count \leftarrow count + 1$
8.     $fcount(v) \leftarrow count$

- Line 3 in Algorithm 1. The complexity of this step is $\times(n)$ (list) or $\mathcal{O}(\log n)$ (heap).
- Line 5 in Algorithm 1 (AddFillEdgesAndRemoveNode). AddFillEdgesAndRemoveNode has three nested loops, each looping over at most all the vertices of the graph. Thus, the complexity of AddFillEdgesAndRemoveNode is $\mathcal{O}(n^3)$.

The complexity of MinFill is dominated by $n$ times the complexity of AddFillEdgesAndRemoveNode, and is thus $\mathcal{O}(n^4)$.

References

Algorithm 3: AddFillEdgesAndRemoveNode(G, v)

Input: A graph $G = (V, E)$, a vertex $v \in V$.
Output: A graph from which $v$ is eliminated and where the fill counts of the remaining vertices are updated.

1. $\text{neigh}[] \leftarrow \text{NEIGHBORS}(v)$  (* Array storing neighbors of $v$ *)
2. for $i \leftarrow 1$ to $\text{SIZE}($neigh$[])$ do
3.   if $\text{fcount}(v) = 0$ then break
4.   $v' \leftarrow \text{neigh}[i]$
5.   for $j \leftarrow i + 1$ to $\text{SIZE}($neigh$[])$ do
6.     if $\text{fcount}(v) = 0$ then break
7.     $v'' \leftarrow \text{neigh}[j]$
8.     if $(v', v'') \notin E$ then
9.       foreach $x \in \text{NEIGHBORS}(v')$ do
10.          if $(x, v'') \in E$ then
11.             $\text{fcount}(x) \leftarrow \text{fcount}(x) - 1$
12.          else
13.             $\text{fcount}(v') \leftarrow \text{fcount}(v') + 1$
14.       foreach $x \in \text{NEIGHBORS}(v'') \land x \neq v$ do
15.          if $(x, v') \notin E$ then $\text{fcount}(v'') \leftarrow \text{fcount}(v'') + 1$
16.     $E \leftarrow E \cup \{(v', v'')\}$
17.   foreach $v' \in \text{NEIGHBORS}(v)$ do
18.     if $\text{fcount}(v') = 0$ then continue
19.     foreach $y \in \text{NEIGHBORS}(v') \land y \neq v$ do
20.        if $(y, v) \notin E$ then
21.           $\text{fcount}(v') \leftarrow \text{fcount}(v') - 1$
22.        if $\text{fcount}(v') = 0$ then break
23. $V \leftarrow V \setminus \{v\}$