REVISED: Implementation of the Minfill Heuristic

Christopher Reeson
Constraint Systems Laboratory
University of Nebraska-Lincoln

Email: creeson@cse.unl.edu

Working Note 1-2015

March 11, 2016

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 partinionning algorithm (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] 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

- Line 1 in Algorithm 1 (FillCount). The complexity of FillCount is determined by the three nested loops: $\mathcal{O}(n^3)$.

[1] http://glaros.dtc.umn.edu/gkhome/views/metis

Algorithm 1: MinFill(G)

```
   Input: A graph G = (V, E), where |V| = n.
   Output: Perfect elimination order σ[]
1  FillCount(G)
2  for i = 1 to n do
3      v ← the vertex in G with the smallest value of fcount
4      σ[i] ← v
5      AddFillEdgesAndRemoveNode(G, v)
6  return σ
```

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
foreach v ∈ V do
    neigh[] ← Neighbors(v)    /* Array storing neighbors of v */
    count ← 0
    for i ← 1 to Size(neigh[]) do
        foreach j ← i + 1 to Size(neigh[]) do
            if (neigh[i], neigh[j]) ∉ E then
                count ← count + 1
    fcount(v) ← 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

[1] Elias Dahlhaus. An improved linear time algorithm for minimal elimination ordering in planar graphs that is parallelizable, 1999.

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.

```
neigh[] ← Neighbors(v)                /* Array storing neighbors of v */
for i ← 1 to Size(neigh[]) do
    if fcount(v) = 0 then break
    v′ ← neigh[i]
    for j ← i + 1 to Size(neigh[]) do
        if fcount(v) = 0 then break
        v″ ← neigh[j]
        if (v′, v″) ∉ E then
            foreach x ∈ Neighbors(v′) do
                if (x, v″) ∈ E then
                    fcount(x) ← fcount(x) − 1
                else
                    fcount(v′) ← fcount(v′) + 1
            foreach x ∈ Neighbors(v″) ∧ x ≠ v do
                if (x, v′) ∉ E then fcount(v″) ← fcount(v″) + 1
            E ← E ∪ {(v′, v″)}
foreach v′ ∈ Neighbors(v) do
    if fcount(v′) = 0 then continue
    foreach y ∈ Neighbors(v′) ∧ y ≠ v do
        if (y, v) ∉ E then
            fcount(v′) ← fcount(v′) − 1
            if fcount(v′) = 0 then break
V ← V \ {v}
```