Implementation of the Minfill Heuristic

Shant Karakashian
Constraint Systems Laboratory
University of Nebraska-Lincoln

Email: shantk@cse.unl.edu

Working Note 1-2010

February 25, 2010

Contents

1 Related Work

The Karypis Lab has developed a large library[1] of graph partitioning 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 An $\mathcal{O}(n^4)$ Implementation for Minfill

Below, we describe an implementation of the well-known minfill algorithm, which is described in [2]. In our implementation, we store in $fcount[x]$ the number of fill edges that need to be added to the graph when the vertex x is removed.

[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 $\sigma[]$

```
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

```
1 foreach v ∈ V do
2 |   count ← 0
3 |   foreach v' ∈ {v'|(v, v') ∈ E} do
4 |   |   foreach v'' ∈ {v''|(v'' ≠ v') ∧ ((v, v'') ∈ E)} do
5 |   |   |   if (v', v'') ∉ E then
6 |   |   |   |   count ← count + 1
7 |   fcount(v) ← count
```

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  foreach v' ∈ {v'|(v, v') ∈ E} do
2      foreach v'' ∈ {v''|(v'' ≠ v') ∧ ((v, v'') ∈ E)} do
3          if (v', v'') ∉ E then
4              foreach x ∈ {x|(x, v') ∈ E} do
5                  if (x, v'') ∈ E then
6                      fcount(x) ← fcount(x) − 1
7                  else
8                      fcount(v') ← fcount(v') + 1
9              foreach x ∈ {x|(x, v'') ∈ E} do
10                 if (x, v') ∉ E then
11                     fcount(v'') ← fcount(v'') + 1
12         E ← E ∪ {(v', v'')}
13 foreach v' ∈ {v'|(v, v') ∈ E} do
14     foreach y ∈ {y|(y ≠ v) ∧ ((y, v') ∈ E)} do
15         if (y, v) ∉ E then
16             fcount(v') ← fcount(v') − 1
17 V ← V \ {v}
```

3 Pseudocode of Cliques and JoinTree

The algorithm for finding the maximal cliques is called Cliques and borrowed from [4].

Algorithm 4: Cliques(G, σ)

Input: Graph $G = (V, E)$ and the perfect elimination order $\sigma[]$, where $|V| = n$.
Output: Q, list of maximal cliques

```
1  Q ← ∅
2  foreach v ∈ V do S(v) ← 0
3  for i = 1 to n do
4      v ← σ(i)
5      X ← {x ∈ Adj(v) | σ^{-1}(v) < σ^{-1}(x)}
6      if Adj(v) = ∅ then Q ← Q ∪ {{v}}
7      if X = ∅ then return Q
8      u ← σ(min{σ^{-1}(x) | x ∈ X})
9      S(u) ← max{S(u), |X| − 1}
10     if S(v) < |X| then
11         Q ← Q ∪ {{v} ∪ X}
12 return Q
```

The algorithm for computing the join tree is the well-known max-cardinality algorithm, also presented in [3].

Algorithm 5: JoinTree(Q)

Input: Q a set of cliques
Output: A join tree $T(Q, E)$

```
1  X ← ∅
2  E ← ∅
3  foreach q ∈ Q do
4      E ← E ∪ {(q, q')} where (q' ∈ X) and (∄q'' ∈ X, |q ∩ q''| > |q ∩ q'|)
5      X ← X ∪ {q}
6  return T(Q, E)
```

4 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: $\mathcal{O}(n^3)$.

- 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 $\mathcal{O}(n^4)$.

The complexity of Cliques is $\mathcal{O}(|V| + |E|)$. The loops iterate $|V|$ times. The call to *Adj* once for each node visits each edge at most twice, hence we add $\mathcal{O}(|E|)$.

The complexity of JoinTree is $\mathcal{O}(|Q|^2)$.

References

[1] Elias Dahlhaus. An Improved Linear Time Algorithm for Minimal Elimination Ordering in Planar Graphs that is Parallelizable, 1999.

[2] Rina Dechter. *Constraint Processing*, chapter Directional Consistency, page 89. Morgan Kaufmann, 2003.

[3] Rina Dechter. *Constraint Processing*, chapter Directional Consistency, page 90. Morgan Kaufmann, 2003.

[4] Martin C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*, chapter Triangulated Graphs, page 99. Academic Press Inc., New York, NY, second edition, 2004.