Minimum Feedback Vertex Set Heuristics

Will Byers
robert_byers@mines.edu

James Crea
jcrea@mines.edu

Bryce Irving
bryceirving@mines.edu

February 28, 2024

Abstract

We present several algorithms for approximating the minimum feedback vertex set problem in polynomial time via simulated automatons, iterative refinement, and divide-and-conquer strategies.

Subroutines

Mapsort

Takes the unordered associative array M as the sole parameter and returns a list of the keys of M sorted in **descending** order by the respective values of M via an efficient (i.e.: $\mathcal{O}(n \cdot \log(n))$) sort algorithm.

Tarjan

Tarjan's strongly connected components algorithm which takes the unweighted, directed, potentially cyclic graph G as the sole parameter and returns a set of subgraphs, such that each strongly connected component c in G maps to a subgraph of G with all of the vertices in c and only the edges within c.

Automaton simulation

function SIMULATE(c, A, S)
    $T \leftarrow$ map from vertices to *traffic*, initialized to 0 for each vertex in c
    for $a = 1, \ldots, A$ **do** ▷ Trivially parallelizable.
        $v \leftarrow$ a random vertex in c
        for $s = 1, \ldots, S$ **do**
            $E \leftarrow$ the set of edges in c for which v is a source vertex
            if length of $E = 1$ **then**
                $v \leftarrow$ the destination vertex of the first and only edge in E
            else
                $v \leftarrow$ the destination vertex of a random edge in E
            end if
            $T[v] \leftarrow T[v] + 1$
        end for
    end for
    $T' \leftarrow$ MAPSORT(T)
    return T'
end function

Where:

Parameter c is a strongly connected, unweighted, directed component.

Parameter A is the number of automatons to spawn.

Parameter S is the number of simulation steps each automaton will run for.

Returned T' is the vertices of c sorted in descending order of traffic

Recursive filter

function RECURSIVE-FILTER(c, T', i, j)
    $L \leftarrow$ empty set of vertices
    function RECURSIVE-FILTER-HELPER(i, j)
        $U \leftarrow$ empty graph
        for vertex $v \in L$ **do**
            Add v and all of its edges in c to U
        end for
        for vertex $v \in T'[i : j)$ **do** ▷ For each vertex in T' from i (inclusive) through j (exclusive).
            Add v and all of its edges in c to U
        end for
        $W \leftarrow$ TARJAN(U)
        if length of $W =$ number of vertices in U **then**
            $L \leftarrow L \cup T'[i : j)$ ▷ Append the vertices in T' from i (inclusive) through j (exclusive) are acyclic.
        else
            $m \leftarrow \lfloor \frac{i+j}{2} \rfloor$
            RECURSIVE-FILTER-HELPER(i, m)
            RECURSIVE-FILTER-HELPER(m, j)
        end if
    end function
    RECURSIVE-FILTER-HELPER(0, $|T'|$)
    return L
end function

Where:

Parameter c is a strongly connected, unweighted, directed component.

Parameter T' is the vertices of c sorted in descending order of traffic.

Parameter i is the start index in T' (inclusive).

Parameter j is the end index in T' (exclusive).

Returned L is the vertices in T' which are acyclic.

Routines

Algorithm 1 One-shot, linear filter heuristic

```
function SOLVE(G, A, S)
    C ← TARJAN(G)
    if length of C = number of vertices in G then
        return empty set of vertices            ▷ G is acyclic, no further processing is necessary.
    end if
    L ← empty set of vertices
    for all strongly-connected component c ∈ C do            ▷ Trivially parallelizable.
        T′ ← SIMULATE(c, A, S)
        U ← empty graph
        for all vertex v ∈ T′ do
            U′ ← U
            Add v and all of its edges in c to U′
            W ← TARJAN(U′)
            if length of W = number of vertices in U′ then
                U ← U′            ▷ Update U with U′ because U′ is acylic.
            end if
        end for
        L ← L ∪ (vertices in c − U)            ▷ Append the vertices in c which are acylic.
    end for
    return L
end function
```

Where:

Parameter G is an unweighted, directed, potentially cyclic graph.

Parameter A is the number of automatons to spawn.

Parameter S is the number of simulation steps each automaton will run for.

Returned L is the (approximately smallest) set of vertices to remove from G to make it acyclic (Note that removing all vertices in L from G is guaranteed to make G acyclic).

Algorithm 2 One-shot, recursive filter heuristic

```
function SOLVE(G, A, S)
    C ← TARJAN(G)
    if length of C = number of vertices in G then
        return empty set of vertices            ▷ G is acyclic, no further processing is necessary.
    end if
    L ← empty set of vertices
    for all strongly-connected component c ∈ C do            ▷ Trivially parallelizable.
        T′ ← SIMULATE(c, A, S)
    end for
    return L
end function
```

Where:

Parameter G is an unweighted, directed, potentially cyclic graph.

Parameter A is the number of automatons to spawn.

Parameter S is the number of simulation steps each automaton will run for.

Returned L is the (approximately smallest) set of vertices to remove from G to make it acyclic (Note that removing all vertices in L from G is guaranteed to make G acyclic).