Listing 1 modified Heuristic Miner

```
Heuristic Miner (EventLog D, Threshold T) {

    #EventLog is sorted based on the TraceIDs and Timestamps
    sort D

    #get number of traces
    def numOfTraces(Eventlog D) {
        for TraceID i in EventLog E do:
            if i = first Element in E do:
                add i to id
            else if i is in id do:
                skip
            else do:
                add i to id
        numTraces = len(id)
    }

    #get transitions
    for Entries i in E do:
        for Entries e in E do:
            if TraceID(i) = TraceID(e) and Activity(e) = Activity(i+1) do:
                add (TraceID(i), Activity(i, e)) to edgesList
        end
    end

    def CountingQuery(edgesList) {
        for Tupel t in edgesList do:
            j := 1
            for Tupel i in edgesList do:
                if t = i and i[TraceID] != i[TraceID] do:
                    count += 1
                else
                    skip
            add count to cQueries
            end
        end
    }

    #filter the transitions and output the filtered list of transitions
         -> which queries are above the threshold
    a := 1
    def AboveThreshold(Database edgesList, Queries cQueries, 70%, $\epsilon_1$){
        while a < n do:
            for Each query i do:
                Let $v_i$ = 70% + Lap(4/$\epsilon_1$)
                if ((numID/numTraces)*100 + $v_i$) $\geq$ 70% do:
                    Output $a_i$ = $\top$
                    Halt.
                else do:
                    Output $a_i$ = $\perp$
                end
            end
            a += 1
        end
    }

    #filter the transitions, so that it is known which queries are below
         the threshold
    b := 1
     def BelowThreshold(Database edgesList, Queries cQuries, 70%, $\epsilon_1$){
          while b < n do:
            for Each query i do:
                Let $v_i$ = 70% + Lap(4/$\epsilon_1$)
                if ((numID/numTraces)*100 + $v_i$) $<$ 70% do:
                    Output $a_i$ = $\top$
                    Halt.
                else do:
                    Output $a_i$ = $\perp$
                end
            end
            b += 1
        end
    }

    #Each query that is $\top$ from AboveThreshold and $\perp$ from BelowThreshold
         is considered, every other is filtered out
    for each Query q from return AboveThreshold:
        for each Query i from return BelowThreshold:
            if q = i and q = $\top$ and i = $\perp$
                add q to consideredQueries

    #cap the frequency of the considered Traces
    def capQuries(cQueries) {
        for Query q in cQueries do:
            if q > cap do:
                q := cap
            add q to cappedQ
    }

    #calculate dependency measure
    def Heuristic(consideredQueries q) {
        for Tripel tl in q do:
            get Tupel i from tl
            x := 0
            if first element of i != last element of i do:
                x = $(|a >^L b| - |b >^L a|)/(|a >^L b| + |b >^L a| + 1)$
                if x >= F do:
                    add (i, x) to heuristicDict
            else do:
                x = $(|a >^L a|)/(|a >^L a| + 1)$
                if x >= F do:
                    add (i, x) to heuristicDict
        end
    }

    #output the directly follows graph
    directly-follows-graph(heuristicDict)

}
```