This is an addendum to

"Strict L_∞ isotonic regression" (2012), *J. Optimization Theory and Applications* 152, pp. 121–135, Quentin F. Stout.

Algorithm C below is a simplified algorithm for determining strict L_∞ isotonic regression of an arbitrary dag. It uses a simple array and a single sort instead of the dynamic priority queue used in Algorithm B. The time constants should be quite small.

If the dag has a transitive closure of m' pairs then the time is $\Theta(m' \log m')$, the same as Algorithm B (this does not count the time to determine the transitive closure, which is part of the input). In terms of the original dag with n vertices this is at most $\Theta(n^2 \log n)$. A more detailed time analysis gives $\Theta(m' + m^\star \log m^\star)$, where $m^\star$ is the number of violating pairs of vertices, i.e., vertices u, v such that $u \prec v$ and $f(u) > f(v)$. The only component taking $\Theta(m^\star \log m^\star)$ time is the sort, with all other lines combined taking only $\Theta(m')$. Thus, for a fixed dag, the more isotonic the data is, the faster the algorithm.

```
input: weighted data (f,w), lists of successors and predecessors for each vertex
output: strict L∞ isotonic regression function S
violators: array of (mean_error,u,v) for violating pairs u ≺ v, f(u) > f(v)
lowbd(v), upbd(v): lower and upper bounds on S(v)

numviolate=0
for every vertex v
     lowbd(v) = −∞;  upbd(v) = +∞;  S(v) = undefined
     for every successor s of v
          if f(v) > f(s) then violators(numviolate)= (mean_err(v,s), v, s);  numviolate++

sort violators by mean_err

for i=0 to numviolate-1
     (mean_err,pred,suc)=violators(i)
     if (S(pred) defined) ∨ (S(suc) defined) then cycle
     wmean = mean(pred,suc)
     if wmean ≥ upbd(pred) then  {f(pred) is ≥ upbd(pred), no later mean is < upbd(pred)}
          S(pred) = upbd(pred)
     if wmean ≤ lowbd(suc) then  {f(suc) is ≤ lowbd(suc), no later mean is > lowbd(suc)}
          S(suc) = lowbd(suc)
     if (S(pred) undefined) ∧ (S(suc) undefined) then  {low(suc) ≤ wmean ≤ high(pred)}
          S(pred) = S(suc) = wmean

     if S(pred) defined then
          for every successor s of pred
               lowbd(s) = max{lowbd(s),S(pred)}
     if S(suc) defined then
          for every predecessor p of suc
               upbd(p) = min{upbd(p),S(suc)}
end for i

for every vertex v
     if S(v) undefined then
          if f(v) ≥ upbd(v) then S(v)=upbd(v)
          else if f(v) ≤ lowbd(v) then S(v)=lowbd(v)
          else S(v)=f(v)
```

Algorithm C: Computing S=Strict(f,w) using transitive closure