Optimizing Recursive Queries with Monotonic Aggregates in DeALS
Alexander Shkapsky, Mohan Yang, Carlo Zaniolo
{shkapsky, yang, zanilo}@cs.ucla.edu
University of California, Los Angeles, Computer Science Department

Datalog Aggregates

Datalog is attracting renewed attention as a tool for many of today’s most challenging computing problems. However, limitations remain for recursive Datalog queries because of the monotonicity requirement upon which the least fixpoint semantics of Datalog is based. In particular, aggregates are needed in many important recursive queries but are disallowed in recursive rules because they are non-monotonic with respect to set-containment.

Efficient Evaluation

In DeALS programs where aggregate values are only operated on by monotonic arithmetic and Boolean expressions, derivations are performed with only max (mmax, mcount, msum) or min (mmin) values. This fact-at-a-time Eager Monotonic Aggregate Semi-naive Evaluation (EMSN) is more efficient than Semi-naive evaluation.

Average for statistic calculated as:
• ≳ 30 trials Counting Paths on each graph size/edge probability combination
• 95% confidence

Analysis for All Pairs Shortest Paths:
• DAGs and Random graphs, 1-50 edge cost
• DAGs – 3-11% more Semi-naive facts
• Random graphs – 13-18% more Semi-naive facts

Max Probability Path

Example: Find max probability path between two nodes in the network.
reach(X,Y,mmax<P>) ← net(X,Y,P).
reach(X,Y,mmax<P>) ← reach(X,Z,P1), reach(Z,Y,P2), P=P1*P2.
maxP(X,Y,mmax<P>) ← reach(X,Y,P).

Shortest Paths

Example: All Pairs Shortest Paths – calculate length of shortest path between each pair of connected nodes in weighted directed graph.
spath(X,Y,mmin<Y>) ← edge(X,Y,C).
spath(X,Y,mmin<Y>) ← spath(X,Z,C1), edge(Z,Y,C2), C=C1+C2.
shortestpath(X,Y,mmin<Y>) ← spath(X,Y,C).

Performance Comparison

DeALS Implementation Details
• Main Memory
• Sequential
• Java

Monotonic Aggregates in DeALS

The Deductive Application Language System (DeALS) supports the \texttt{mmax}, \texttt{mmin}, \texttt{mcount} and \texttt{msum} aggregates that are monotonic and continuous in the lattice of set-containment and can be used in recursion. DeALS achieves great expressive power by supporting these new monotonic constructs along with various \texttt{DL++} non-monotonic constructs, such as choice and XY-stratification.

\textbf{Example:} Number of children for a parent.
cntchild(X,mcount<(C,1)>) ← parent(X,C).
This produces: \texttt{cntchild(eve,1)}.
\texttt{cntchild(eve,2)}.
\texttt{cntchild(eve,3)}.

We need a stratified program for the actual max.
\texttt{exactCnt(X,max<Y>)} ← \texttt{cntchild(X,Y)}.
This produces: \texttt{exactCnt(eve,3)}.

\textbf{mcount}

Example: There is a party on campus, and a student will attend the party if at least three of their friends will attend. Organizers attend.
cntfriends(Y,mcount<(X,1)>) ← attend(X), friend(Y,X).
attend(Y) ← organizer(Y).
attend(Y) ← cntfriends(Y,N), N >= 3.
\textbf{Intuition:} This program is monotonic -> additions to \texttt{cntfriends} and attend will only cause attend to grow.
organizer(tom), organizer(sue), organizer(pat).
friend(marc,tom), friend(marc,sue), friend(marc,pat).
friend(ann,tom), friend(ann,pat), friend(ann,marc).
\textbf{Results:} The organizers tom, sue and pat will attend; Then marc, will attend. Finally ann, will also attend.

Example: Count the number of paths between two nodes in a directed acyclic graph.
cpath(X,Y,mcount<(X,1)>) ← edge(X,Y).
cpath(X,Y,mcount<(Z,1)>) ← cpath(X,Z,C), edge(Z,Y).
countpaths(X,Y,max<Z>) ← cpath(X,Y,C).

Counting the occurrences of \texttt{(Z,C)} is the same as adding up \texttt{max<Z>} for each \texttt{Z}.

\textbf{BOM}

\textbf{Example:} Find maximum cost of a part in an assembly.
\texttt{cost(P,msum<(P,C)>)} ← basic(P,C).
\texttt{cost(P,msum<(SP,C)>)} ← assbl(P,SP,N), \texttt{cost(SP,SC)}, C=SC*N.
\texttt{totalCost(P,max<Z>)} ← \texttt{cost(P,C)}.

Conclusion

• Monotonic Aggregates in DeALS are uniquely powerful.
• The aggregates are amenable to highly efficient evaluation.
• DeALS implementation provides good overall performance. Parallelization is the next challenge!

\texttt{http://wis.cs.ucla.edu/deals}