Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products

Guido Moerkotte   Thomas Neumann

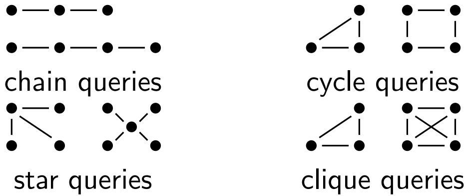
September 15, 2006

Overview

1. Motivation
2. Existing Algorithms: DPsize, DPsub
3. Idea
4. Our Algorithm: DPccp
5. Evaluation
6. Conclusion

Motivation

Problem: Generate the best bushy join tree not containing a cross product.



- structure of query graph greatly affects complexity
- e.g. cliques are NP hard in general, chains are in $O(n^3)$
- algorithm should adapt to the graph structure

Motivation - Dynamic Programming Strategies

Advantages:

- general purpose, many cost functions
- find the optimal solution

Basic scheme:

- solve problems only once
- build solutions from smaller solutions
- here: join pairs of optimal join trees
- main difference between strategies: enumeration order

query graph structure should affect enumeration order

Existing Algorithms - DPsize

- organize DP by the size of the join tree
- enumerate ordered by the number of joined relations
- first all with 2 relations, with 3 relations, etc.
- for a given size n consider all L, R such that $n = |L| + |R|$
- prune pairs afterwards (connectedness, disjointness, costs)
- problem: only few DP slots, many pairs considered

good algorithm for chains, very bad for cliques:

	chains	cycles	stars	cliques
pairs	$O(n^4)$	$O(n^4)$	$O(4^n)$	$O(4^n)$

absolute complexity also interesting, see the paper

Existing Algorithms - DPsub

- organize DP by the set of relations involved
- enumerate subsets before supersets
- first $\{R_1\}$, then $\{R_2\}$, then $\{R_1, R_2\}$ etc.
- for a given problem P consider all L,R such that $P = L \cup R, L \cap R = \emptyset$
- prune pairs afterwards (connectedness, costs)
- problem: always 2^n DP slots, fixed enumeration

good algorithm for cliques, but adapts badly:

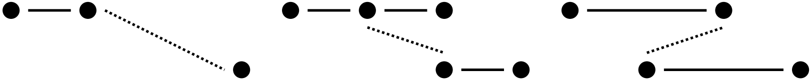
	chains	cycles	stars	cliques
pairs	$O(2^n)$	$O(n2^n)$	$O(3^n)$	$O(3^n)$

faster than DPsize for stars and cliques, slower for chains and cycles.

Idea - Observation

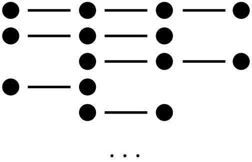
DPsize and DPsub generate many pairs that are pruned anyway (connectedness, overlap).

Typical pruned pairs (chain with 4 relations):



not connected — not disjoint — invalid subproblems

last example ⇒ every join partner must be a connected subgraph:



Idea - New Approach

- reformulation as graph theoretic problem:
- enumerate all connected subgraphs of the query graph
- for each subgraph enumerate all other connected subgraphs that are disjoint but connected to it
- each connected subgraph - complement pair (ccp) can be joined
- enumerate them suitable for DP $\Rightarrow$ DP algorithm

algorithm adapts naturally to the graph structure:

	chains	cycles	stars	cliques
pairs	$O(n^3)$	$O(n^3)$	$O(n2^n)$	$O(3^n)$

Lohman et al: #ccp is a lower bound for all DP enumeration algorithms

Idea - Effect on Search Space

Absolute number of generated pairs

	Chain			Star		
n	#ccp	DPsub	DPsize	#ccp	DPsub	DPsize
2	1	2	1	1	2	1
5	20	84	73	32	130	110
10	165	3,962	1,135	2,304	38,342	57,888
15	560	130,798	5,628	114,688	9,533,170	57,305,929
20	1,330	4,193,840	17,545	4,980,736	2,323,474,358	59,892,991,338
	Cycle			Clique		
n	#ccp	DPsub	DPsize	#ccp	DPsub	DPsize
2	1	2	1	1	2	1
5	40	140	120	90	180	280
10	405	11,062	2,225	28,501	57,002	306,991
15	1,470	523,836	11,760	7,141,686	14,283,372	307,173,877
20	3,610	22,019,294	37,900	1,742,343,625	3,484,687,250	309,338,182,241

New Algorithm

- two steps: enumerate all connected subgraphs, enumerate disjoint but connected subgraphs for a given one $\Rightarrow$ pairs
- enumerate all pairs, enumerate no duplicates, enumerate for DP
- if (a, b) is enumerated, do not enumerate (b, a)
- requires total ordering of connected subgraphs
- preparation: label nodes breadth-first from 0 to $n-1$

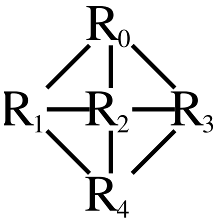
Preliminaries, given query graph $G = (V, E)$:

$$\begin{aligned} V &= \{v_0, \ldots, v_{n-1}\} \\ \mathcal{N}(V') &= \{v' | v \in V' \wedge (v, v') \in E\} \\ \mathcal{B}_i &= \{v_j | i \leq i\} \end{aligned}$$

New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \dots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

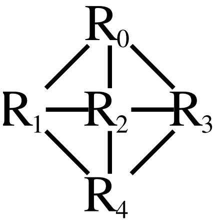


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

Choose all nodes as enumeration start node once

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

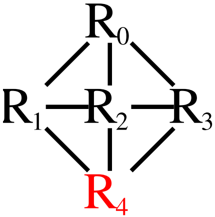


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

First emit only the node itself as subgraph

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

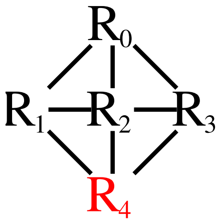


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

Then enlarge the subgraph recursively

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

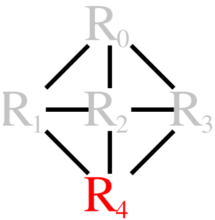


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
  emit $\{v_i\}$;
  EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

Prohibit nodes with smaller labels. Thus the set of valid nodes increases over time

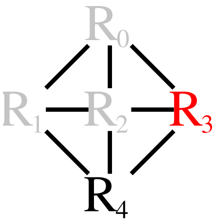
EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
  emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
  EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}



New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

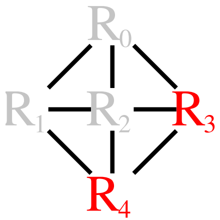
EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}



New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** $\{$
  emit $\{v_i\}$;
  EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
$\}$

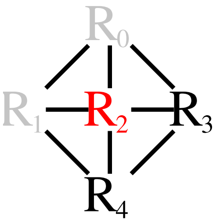
EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first $\{$
  emit $(S \cup S')$;
$\}$
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first $\{$
  EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
$\}$



New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

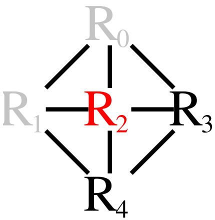


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

In each recursion, find all neighboring nodes that are not prohibited

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

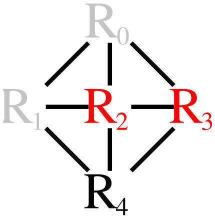


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
  emit $\{v_i\}$;
  EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

Add all combinations to the subgraph and emit the new subgraph

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
  emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
  EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

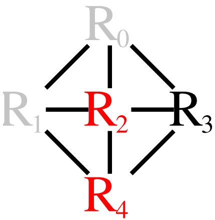


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

Add all combinations to the subgraph and emit the new subgraph

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

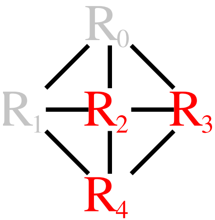


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
  emit $\{v_i\}$;
  EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

Add all combinations to the subgraph and emit the new subgraph

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
  emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
  EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

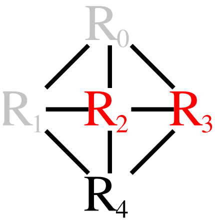


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

Then, add all combinations to the subgraph and increase recursively

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}

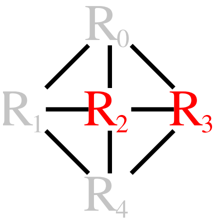


New Algorithm - Connected Subgraphs

EnumerateCsg(G)
for all $i \in [n-1, \ldots, 0]$ **descending** {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $\mathcal{B}_i$);
}

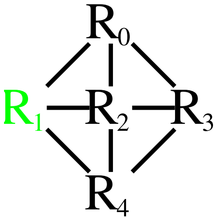
The neighborhood is prohibited during recursion, preventing duplicates

EnumerateCsgRec(G, S, X)
$N = \mathcal{N}(S) \setminus X$;
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    emit $(S \cup S')$;
}
for all $S' \subseteq N$, $S' \neq \emptyset$, enumerate subsets first {
    EnumerateCsgRec(G, $(S \cup S')$, $(X \cup N)$);
}



New Algorithm - Complementary Subgraphs

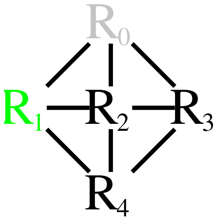
EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}



New Algorithm - Complementary Subgraphs

EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}

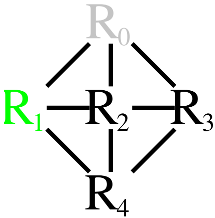
Prohibit all nodes that will be start nodes later on and the primary subgraph



New Algorithm - Complementary Subgraphs

EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}

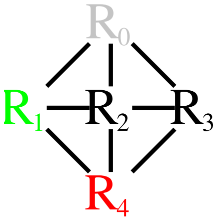
Find all neighboring nodes that are not prohibited



New Algorithm - Complementary Subgraphs

EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}

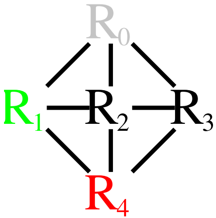
Consider each of the nodes



New Algorithm - Complementary Subgraphs

EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}

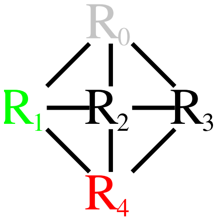
Choose the node as complementary subgraph and emit it



New Algorithm - Complementary Subgraphs

EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}

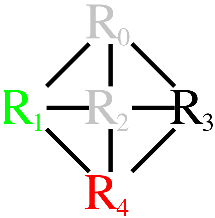
Recursively increase the subgraph re-using EnumerateCsgRec



New Algorithm - Complementary Subgraphs

EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
  emit $\{v_i\}$;
  EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}

Again prohibit nodes with a smaller label to prevent duplicates



New Algorithm - Complementary Subgraphs

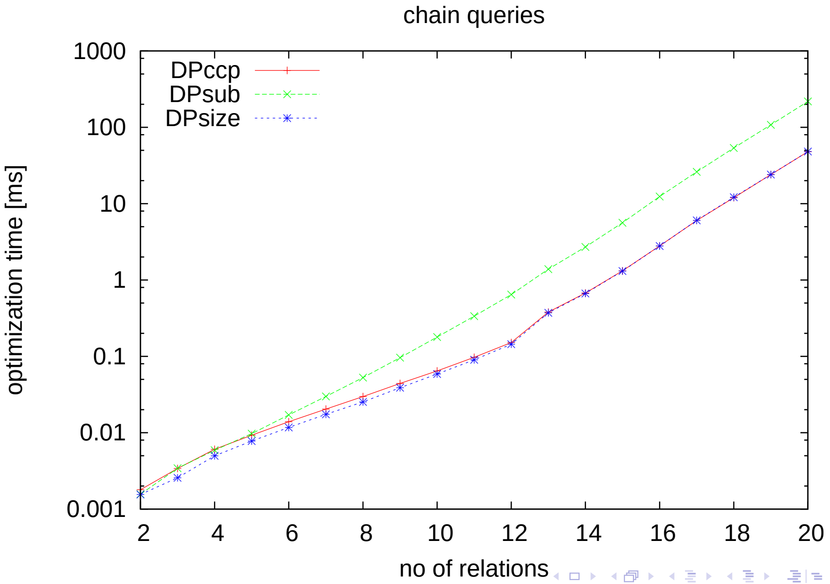
EnumerateCmp(G,S_1)
$X = \mathcal{B}_{\min(S_1)} \cup S_1$;
$N = \mathcal{N}(S_1) \setminus X$;
for all ($v_i \in N$ by descending i) {
    emit $\{v_i\}$;
    EnumerateCsgRec(G, $\{v_i\}$, $X \cup (\mathcal{B}_i \cap N)$);
}

- EnumerateCsg+EnumerateCmp produce all ccp
- resulting algorithm DPccp considers exactly #ccp pairs
- which is the lower bound for all DP enumeration algorithms
- formal proof of correctness in the paper

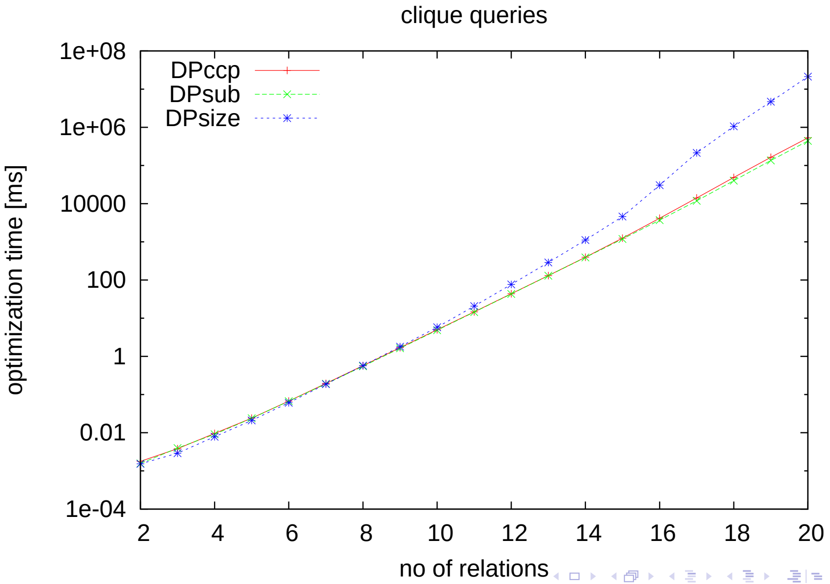
Evaluation

- asymptotically DPccg is clearly superior
- but implementation is more involved
- measure overhead by comparing runtime
- extremes: chain (favors DPsize) and clique (favors DPsub)
- in between: stars, show effect of search space reduction
- real queries will also be between chains and cliques

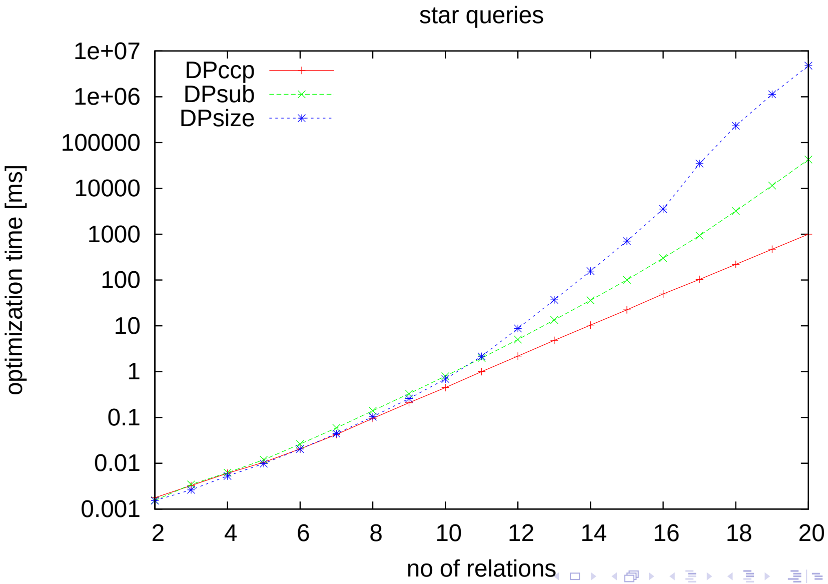
Evaluation - Chains



Evaluation - Cliques



Evaluation - Stars



Conclusion

- analytic and experimental evaluation of DPsize/DPsub
- DPsize is superior for chains/cycles
- DPsub is superior for stars/cliques
- new algorithm DPccg adopts to query graph structure
- minimal number of pairs
- low implementation overhead
- DPccp is the DP algorithm to choose

Number of Connected Subgraphs

	chains	cycles	stars	cliques
#csg	$O(n^2)$	$O(n^2)$	$O(2^n)$	$O(2^n)$

- determines the size of the DP table
- determines the number of cardinality estimations
- much less than #ccp