MiniCon

Answering Queries Using Views: A Survey. Alon Y. Halevy

Stanislav Angelov, CIS650 Data Sharing and the Web
U. of Pennsylvania

Additional references: MiniCon: A Scalable algorithm for answering queries using views. R. Pottinger, A. Halevy

Inverse-rules Algorithm

Construct set of rules that invert the view definition.

Pros	Cons
■ Simplicity and modularity ■ Returns maximally contained rewriting even w/ arbitrary recursive Datalog programs	■ May invert some of the useful computation done to produce view
■ Needs additional constant propagation to trim redundant computations	■ Needs additional constant propagation to trim redundant computations

Bucket Algorithm

1) Create a bucket for each sub-goal in the query containing the views that have the same sub-goal and there is a mapping. 2) Consider conjunctive rewriting for each element of the Cartesian product of the buckets, and check whether it is contained or can be made to be contained in the query.

Pros	**Cons**
Considers each sub-goal in isolation	*Considers each sub-goal in isolation*

Pros

- To some degree takes into account context to prune search space
- Would possibly take advantage of materialized views

Cons

- Considers Cartesian product of buckets
- It is hard to recover projected away attributes w/o additional knowledge

MiniCon Algorithm

- Inverse-rules algorithm (extended version) is very similar to the Bucket algorithm and performs better.
- ***Main objective of the MiniCon Algorithm is to scale better with the number of available views***
- **Key difference between MiniCon and the above algorithms is the MiniCon Descriptors computed for each goal mapping**

- **more preprocessing to build Descriptors (scale nicely to number of goals/views)**

- **less work on combining phase (potentially exponential).**

MiniCon Algorithm Outline

- 1a) Begin like the Bucket Algorithm
- 1b) Form the MiniCon Descriptors

  For sub-goal g in the query Q mapped to sub-goal g' in view V (bucket), look at the variables Q and consider the join predicates to find the minimal additional set of sub-goals in Q that must be mapped to sub-goal in V in order V be usable.
- 2) Combine MCD-s

- proceed as in the bucket algorithm but consider rewritings involving only disjoin MCD-s

- no need of containment check (additional speedup) for each rewriting

Example: Setting and 1st phase

- q(D) :- Major(S, D), Registered(S, 444, Q), Advises(P, S)

V1(dept) :- Major(student,dept), Registered(student,444,quarter)

V2(prof,student,area) :- Advises(prof,student), Prof(prof,area)

V3(dept,c-number) :- Major(student,dept), Registered(student,c-number,quarter), Advises(prof,student)

Bucket Algorithm (phase 1):

1. Major(S,Q)	2. Registered(S,C,Q)	3. Advises(P,S)
V1(D’)	V1(D’)	
		V2(P,S,A’)
V3(D’,C’)	V3(D’,C)	V3(D’,C’)

Example: MCD Construction 1/3

- q(D) :- Major(S, D), Registered(S, 444, Q), Advises(P, S)

V1(dept) :- Major(student,dept), Registered(student,444,quarter)

1. Major(S,Q)	2. Registered(S,C,Q)	3. Advises(P,S)
~~V1(D')~~	~~V1(D')~~	
		V2(P,S,A')
V3(D',C')	V3(D',C)	V3(D',C')

- In order V1 (q:Major -> V1:Major) to be usable we need to be able to join Major with Registered and Advises on Student. Since Student is not in the head of V1, V1 should include those two joins but is include only of them.
- So join with Advises on S cannot be done unless additional functional dependencies exist and are known.
- We apply the same argument to determine that the mapping (q:Registered -> V1:Registered) is not possible.

Example: MCD Construction 2/3

- q(D) :- Major(S, D), Registered(S, 444, Q), Advises(P, S)

V2(prof,student,area) :- Advises(prof,student), Prof(prof,area)

1. Major(S,Q)	2. Registered(S,C,Q)	3. Advises(P,S)
~~V1(D')~~	~~V1(D')~~	
		V2(P,S,A')
V3(D',C')	V3(D',C)	V3(D',C')

- In order V2 (q:Advises->V2:Advises) to be usable we need to be able to join it with Major and Registered on Student. Since (S-> Student) is in the head of V2 we can apply the join predicates later and we don't need to do additional mappings.

Example: MCD Construction 3/3

- q(D) :- Major(S, D), Registered(S, 444, Q), Advises(P, S)

V3(dept,c-number) :- Major(student,dept),
Registered(student,c-number,quarter),
Advises(prof,student)

1. Major(S,Q)	2. Registered(S,C,Q)	3. Advises(P,S)
~~V1(D’)~~	~~V1(D’)~~	
		V2(P,S,A’)
V3(D’,C’)	V3(D’,C)	V3(D’,C’)

- In order V3 (q:Major->V3:Major) to be usable we need to be able to join it with Registered and Advises on Student. Since S is not in the head of V3 we have to map Registered and Advises also.

Example: Combining MCD-s

- q(D) :- $\overbrace{\text{Major(S, D)}}^{1}$, $\overbrace{\text{Registered(S, 444, Q)}}^{2}$, $\overbrace{\text{Advises(P, S)}}^{3}$
- MCD-s

V(Y)	Head homomorphism* h	φ**	Sub-goals
~~V1(dept)~~			~~1,2,3~~
V2 (prof,student,area)	identity	P $\rightarrow$ profs, S $\rightarrow$ students	3
V3 (dept,c-number)	identity	D->dept c-number->444	1,2,3

* Can equate distinguished variables h(x)=h(h(x))

** Partial mapping of Vars(Q) to head h(Vars(V))

Now we have to consider only disjoined MCD-s when combining. V2's and V3's are not disjoined so we would consider only rewriting involving V3

Rules and Properties of MiniCon

- **For a query Q sub-goal g, and a view V sub-goal g', we map g to g', with the following properties for every query variable X that is mapped to view variable A:**
- Case I: X is head variable, A is head variable OK
- Case II: X is not head variable, A is head variable OK
- Case III: X is head variable, A is not head variable NOT OK

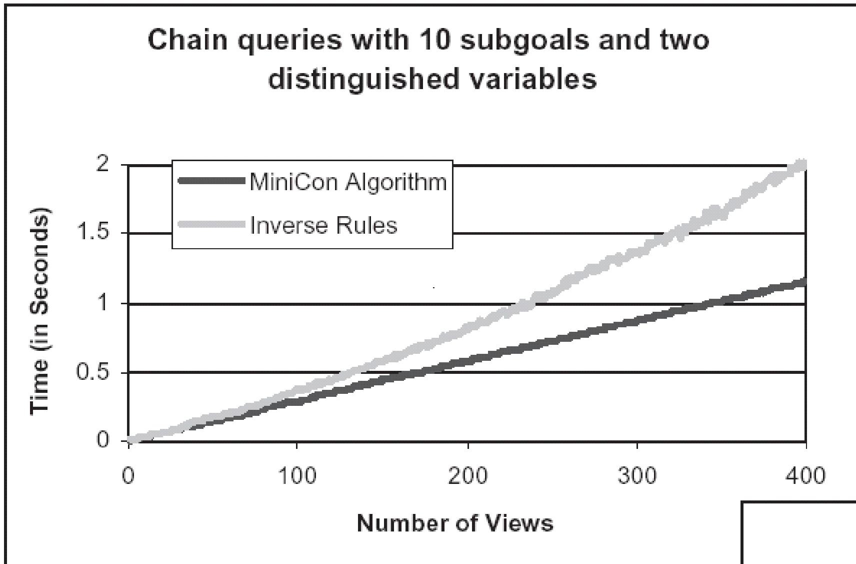
(x need to be in the answer but a is not exported)

- Case IV: X is not head variable, A is not head variable ???

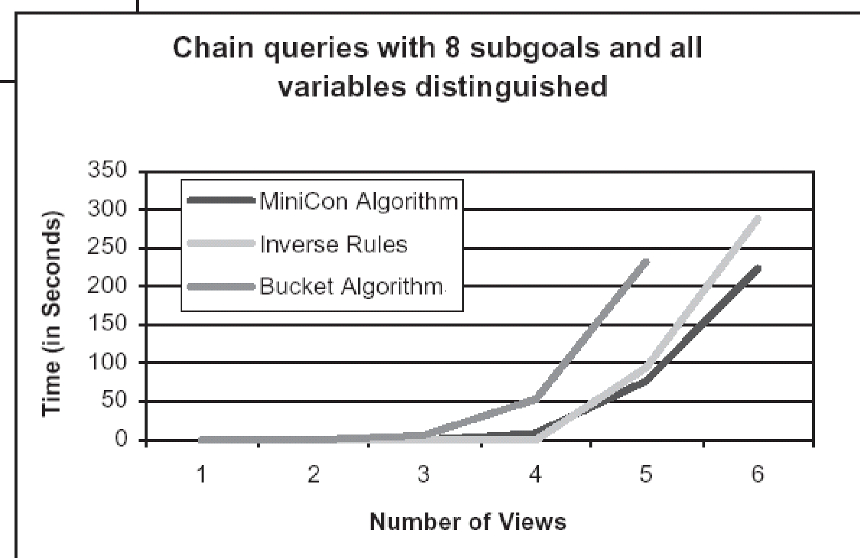
All the query sub-goals using X must be able to be mapped to other sub-goals in V in order to be able to reconstruct the join

- **Given a query Q, a set of views V, and the set of MCD-s C for Q over the views in V, the only combinations of MCD-s that result in non-redundant rewritings of Q are of the from C_1, C_2, …, $C_{l,\ where}$**
- Sub-goals(Q) = Goals(C_1) $\cup$ Goals(C_2) $\cup$ …
- For every $i \neq j$, Goals(C_i) $\cap$ Goals(C_j) = $\varnothing$

Experimental Results

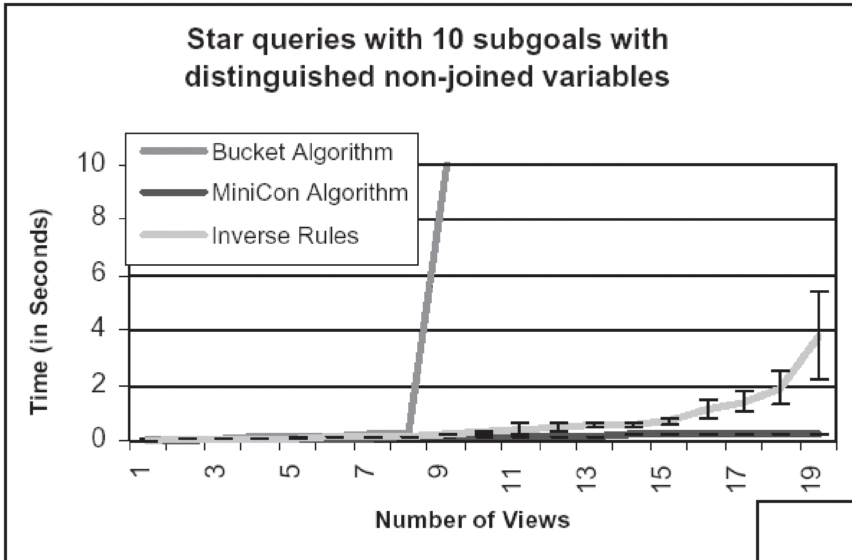


- Chain queries with only few rewritings

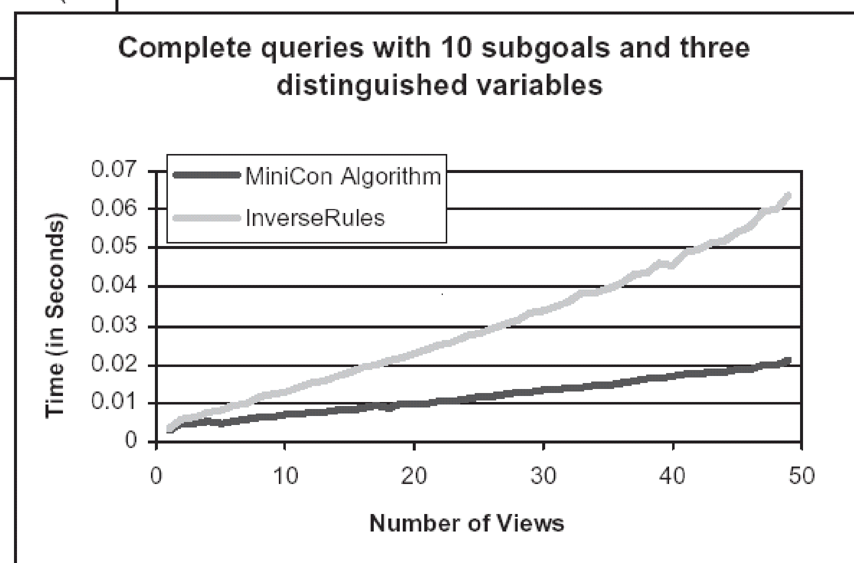


- Chain queries with many rewritings

Experimental Results

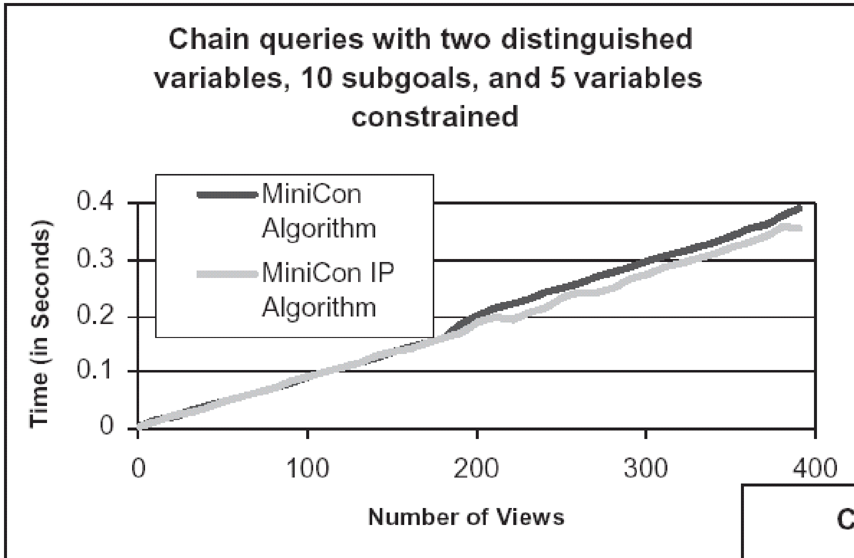


- Unique sub-goal joins with everything. Few rewritings.

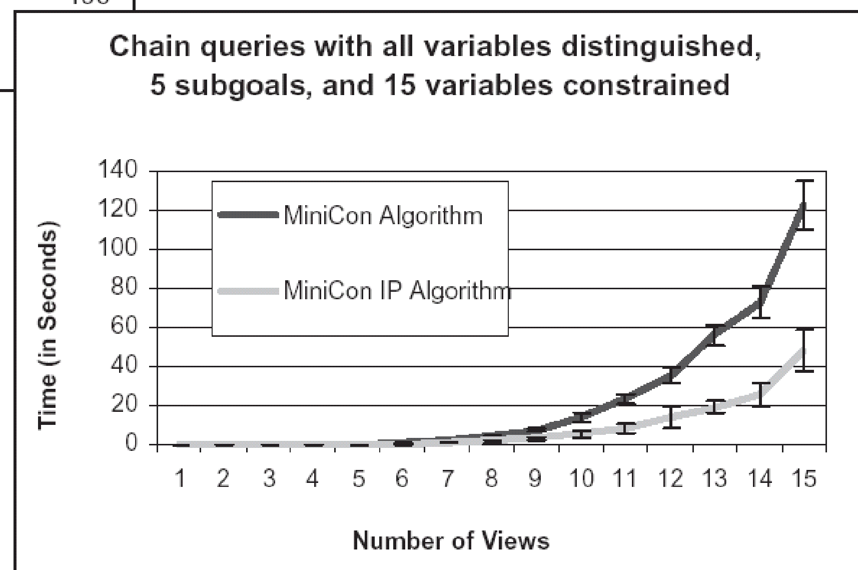


- Every sub-goal is joined with every other sub-goal. Prune earlier.

Experimental Results



- Comparison predicates don't slow MiniCon too much. Only few rewritings so overhead not large.



- Prune rewritings because of comparison predicates

More and Conclusions

- Completeness
- Certain answers and maximally-contained rewritings (closed-world, open world assumptions)
- Use of MiniCon in the context of query optimization
- MiniCon algorithm scales better with the number of views. Though it requires more preprocessing, it reduces the work at the more expensive rewriting phase