Automatic Index Selection for Inequalities

Samuel Isaac Arch

Supervisor: Bernhard Scholz

THE UNIVERSITY OF SYDNEY

Introduction

- Datalog is popular for its conciseness and expressiveness [1]
  - Static Program Analysis (DOOP)
  - Network Analysis (VPC)
  - Binary Disassembly (DDISASM)
- Competitive with hand-crafted tools at giga-scale (SOUFFLÉ) [2]
- Current evaluation of inequalities fails to meet real-world demands (DDISASM)

We present two new approaches to speed up inequalities automatically

Datalog

Declarative programming language - specify the logic of a computation using rules

Rules

```
path(x, y) :- edge(x, y).
path(x, z) :- edge(x, y), path(y, z).
```

Facts (Input)

```
edge(1, 2), edge(2, 3).
```

Knowledge (Output)

```
path(1, 2), path(2, 3), path(1, 3).
```

Evaluating Datalog with Soufflé

Logic programs are transformed into equivalent imperative programs

Original Rule

```
path(x, z) :- edge(x, y), path(y, z).
```

Transformed Rule

```
path(a, d) :- edge(a, b), path(c, d), c = b.
```

Loop Nest

for all $t_0 \in edge$ **do**
  for all $t_1 \in path$ **do**
    if $t_1(c) = t_0(b)$ **do**
      if $(t_0(a), t_1(d)) \notin path$ **do**
        project $(t_0(a), t_1(d))$ **into** $path$

+ No need to materialise intermediate relations
– Complexity proportional to the size of the Cartesian product of involved relations

Equality Primitive Searches

An *equality primitive search* [3] is a filter operation on a Datalog relation where a subset of the relation's attributes are equal to constant values i.e.

$$\sigma_{x_1=v_1,...,x_k=v_k}(R) = \{t \in R \mid t(x_1) = v_1, ..., t(x_k) = v_k\}$$

Key Idea

Hoist equality predicates on relations to make equality primitive searches

Table Scan and Filter

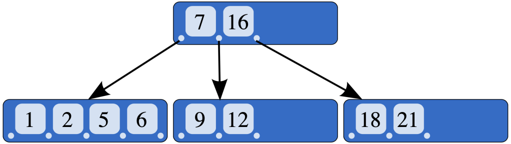
for all $t_0 \in edge$ **do**
  for all $t_1 \in path$ **do**
    if $t_1(c) = t_0(b)$ **do**
      if $(t_0(a), t_1(d)) \notin path$ **do**
        project $(t_0(a), t_1(d))$ **into** $path$

Equality Primitive Search

for all $t_0 \in edge$ **do**
  for all $t_1 \in \sigma_{c\,=\,t_0(b)}(path)$ **do**
    if $(t_0(a), t_1(d)) \notin path$ **do**
      project $(t_0(a), t_1(d))$ **into** $path$

Indexes

Storing logical relations in data structures called *indexes* can accelerate searches

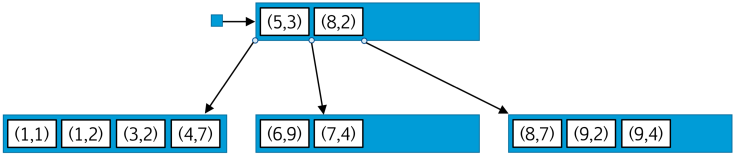


B-Trees

- \+ Complexity of evaluating searches is bounded by the size of the output i.e. $\mathcal{O}(\log(n) + |Q|)$
- \+ Tree structures provide natural opportunities for parallelism
- \+ Effectively exploits caches available
- – Not designed to store multi-dimensional data (i.e. tuples)

Indexing Multi-Dimensional Data with B-Trees

B-Trees are uni-dimensional and require a total ordering of all tuples



Lexicographical Ordering

- Imposes a total ordering on the set of tuples by comparing attributes one at a time
- Provides support for all standard B-Tree operations
- For the above example we use the lex-order $\ell = x_1 \prec x_2$

Which B-Tree Indexes do we Build?

Selecting the right indexes is tricky

Building a Single B-Tree Index

+ Speeds up some of the searches
– Some searches may be uncovered causing dramatic performance degradation

Building Multiple B-Tree Indexes

+ Speeds up every search for a relation
– Replica indexes require maintenance

The Minimum Index Selection Problem (MISP) [3]

Given a collection of search sets $\mathcal{S}$ on a relation R, compute the minimum cardinality set of B-Tree indexes that cover every search.

Notation

- Search sets abstract equality primitive searches since constants are not important
- For example: $\sigma_{x=2,\ y=3,\ z=5} \mapsto \{x, y, z\}$

Rationale

- Minimise the maintenance cost while still covering every search

Solving the MISP via Minimum Chain Covering [3]

Key Idea

How do we find an index ℓ for two searches S and S'?

An index $\ell = S \prec (S' - S)$ covers both if $S \subset S'$

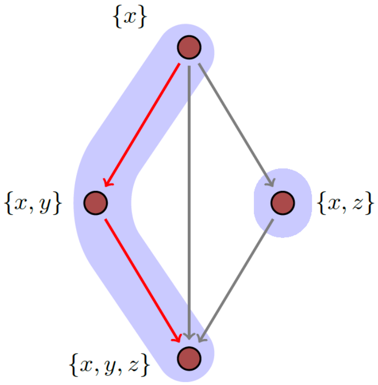
e.g. $\ell = x \prec y$ covers $S = \{x\}$ and $S' = \{x, y\}$

What if we have a chain of searches?

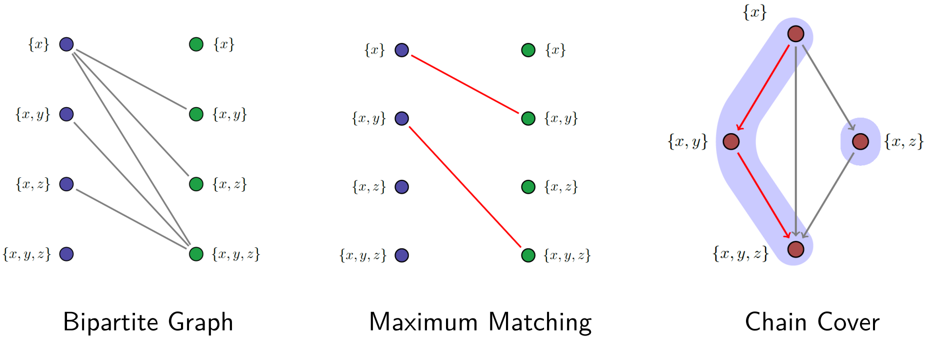
$C = S_1 \subset S_2 \subset ... \subset S_k$

There is always an index to cover the whole chain

$\ell = S_1 \prec (S_2 - S_1) \prec ... \prec (S_k - S_{k-1})$



Minimum Chain Covering by Dilworth's Theorem



(1) Construct a bipartite graph from $\mathcal{S} \times \mathcal{S}$

(2) Draw an edge from S to S' if $S \subset S'$

(3) Compute the maximum cardinality matching on the graph

(4) Map edges from the matching to the minimum chain cover

(5) Create indexes for each chain $\ell = S_1 \prec (S_2 - S_1) \prec \ldots \prec (S_k - S_{k-1})$

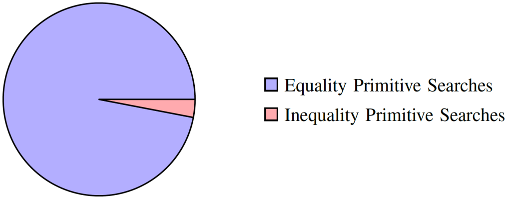
Automatic Index Selection in Soufflé

The state-of-the-art auto-index selection technique is currently deployed in Soufflé

Summary

+ Entirely automatic (no user intervention)
+ Polynomial time index selection (negligible compilation time overhead)
+ Robust as every equality primitive search is covered by an index
− Searches with *inequalities* are not covered by an index (unacceptable performance)

Spatial Primitive Searches



A *spatial primitive search* is a filter operation on a Datalog relation where a subset of the relation's attributes are lower bounded and/or upper bounded by constant values i.e.

$$\sigma_{l_1 \leq x_1 \leq u_1, \ldots, l_k \leq x_k \leq u_k}(R) = \{t \in R \mid l_1 \leq t(x_1) \leq u_1, \ldots, l_k \leq t(x_k) \leq u_k\}$$

- An SPS is an equality primitive search when for all x_i we have $l_i = u_i$
- An SPS is an inequality primitive search otherwise
- The SPS gadget distils each Datalog operation down to its semantics

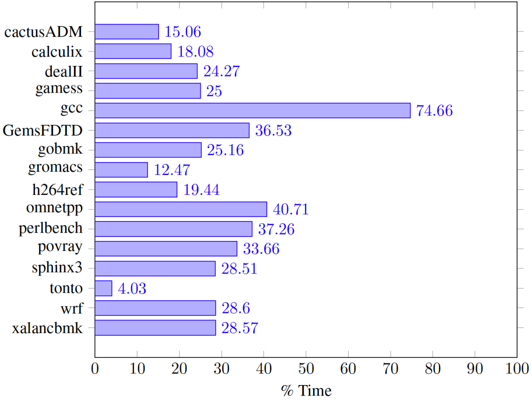
Statistics on Inequality Primitive Searches

Proportion of Program (%)

- DOOP (4.46%)
- VPC (4.9%)
- DDISASM (3.29%)

Proportion of Evaluation Time (%)

- DOOP (Less than 1.29%)
- VPC (Less than 0.01%)
- DDISASM (Up to 74.66%)



We consider rules which contain at least one inequality primitive search

How Do We Speed up Inequalities?

The only literature on inequalities in Datalog is over 8 years old [4]

Existing Technique

- Program level transformation
- Creates new "Filter Relations" that are smaller than the original relations
- (1) `c(x) :- a(x), b(y), y < x.`
- (2a) `c(x) :- a(x), b_filtered(y), y < x.`
- (2b) `b_filtered(y) :- b(y), y < max x : { a(x) }.`

DDISASM Rule (58% of Evaluation Time)

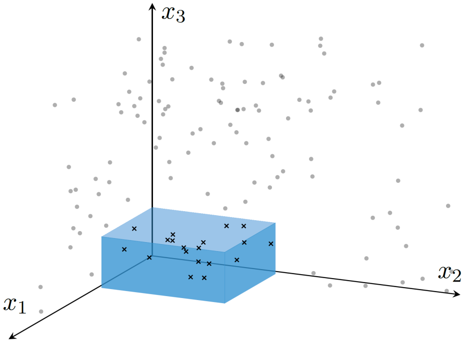
```
data_object_conflict(EA1, Size1, Type1, EA2, Size2, Type2) :-
     data_object_candidate(EA1, Size1, Type1),
     data_object_candidate(EA2, Size2, Type2),
     EA1 < EA2, EA2 < EA1 + Size1
```

Spatial Primitive Search $\longleftrightarrow$ Orthogonal Range Query

Orthogonal range querying is the problem of finding from a set of d-dimensional points S, a subset Q of points that lie within a specified d-dimensional box

Reduction

- Given an SPS: $\sigma_{l_1 \leq x_1 \leq u_1, \ldots, l_k \leq x_k \leq u_k}(R)$
- Define the bounding box: $B = b_1 \times \ldots \times b_d$
- Set $b_i = [lower_i, upper_i]$
- If l_i is specified, $lower_i = l_i$ otherwise $lower_i = \inf(\mathbb{D}_i)$
- If u_i is specified, $upper_i = u_i$ otherwise $upper_i = \sup(\mathbb{D}_i)$
- Easy to prove that the semantics coincide



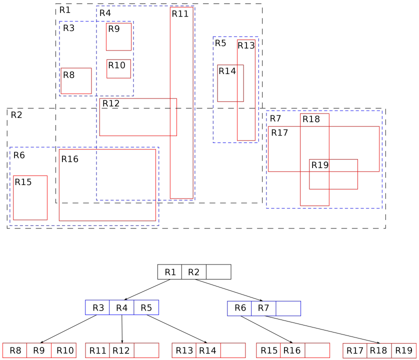
Therefore, a single index supporting orthogonal range queries covers every SPS of a relation

Which Index for Orthogonal Range Querying?

Datalog involves large amounts of updating indexes as new knowledge is derived

Rationale for R-Trees [5]

- Kd-Trees/Range Trees are not dynamic
- Dynamic Range Trees fail in practice [6]
- R-Trees are ubiquitous in spatial databases (MySQL, PostGIS, Oracle Spatial etc.)
- R* variant offers fast query performance (although worst case complexity is $\mathcal{O}(n)$)



R-Tree SPS

- Use a single R-Tree index for relations with inequality primitive searches

Extending the State of the Art

The original auto-index technique with a cluster of B-Trees performs well, so why not extend it?

Problem: B-Trees are Uni-dimensional Structures

- B-Trees cannot perform range queries over more than 1 dimension
- At most 1 attribute can have an inequality constraint i.e. $l_i \neq u_i$ for some x_i
- A *simple spatial primitive search* is an SPS which satisfies the above condition

Single Search and Filter

for all $t_0 \in R_0$ **do**

...

for all $t_k \in \sigma_{l_1 \leq x_1 \leq u_1}(R_k)$ **do**

if $l_2 \leq x_2 \leq u_2$ **and** $l_3 \leq x_3 \leq u_3$ **do**

...

project (...) **into** ...

The MISP for Simple Spatial Primitive Searches

Given a collection of search set pairs $(S_{EQ}, S_{INEQ}) \in \mathcal{S}$ satisfying $|S_{INEQ}| \leq 1$ on a relation R, compute the minimum cardinality set of B-Tree indexes that cover every search set pair.

Notation

- Pair of search sets (S_{EQ}, S_{INEQ})
- S_{EQ} contains all constrained attributes where $l_i = u_i$
- S_{INEQ} contains all constrained attributes where $l_i \neq u_i$
- For example: $\sigma_{x=2,\ y<3,\ z=5} \mapsto (\{x, z\}, \{y\})$
- We define $S = S_{EQ} \cup S_{INEQ}$ to contain all attributes in the search

B-Tree SPS

- Use the smallest possible cluster of B-Trees to cover every simple SPS

Solving the MISP via Minimum Chain Covering [3]

Key Idea

Define a new partial order over search set pairs

We write $(S_{EQ}, S_{INEQ}) < (S'_{EQ}, S'_{INEQ})$ if:

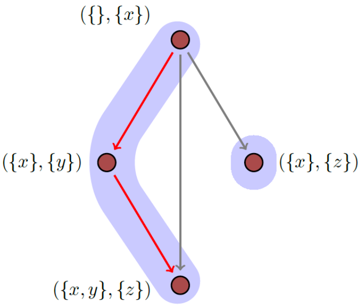
(1) $S \subseteq S'$ i.e. $(S_{EQ}, S_{INEQ}) \subseteq (S'_{EQ}, S'_{INEQ})$
(2) if $x_i \in S'_{INEQ}$ then $x_i \notin S$

For any chain of search set pairs

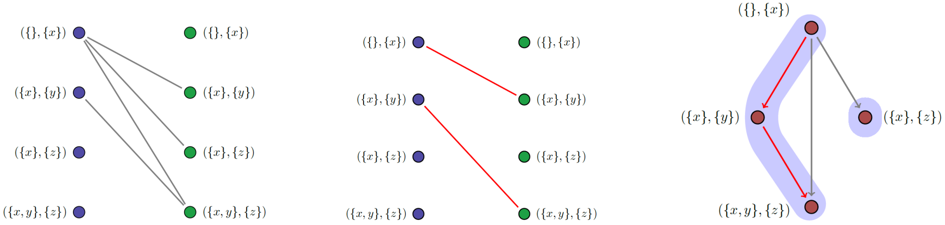
$$C = (S^1_{EQ}, S^1_{INEQ}) < ... < (S^k_{EQ}, S^k_{INEQ})$$

There is always an index to cover the whole chain

$$\ell = S^1 \prec (S^2 - S^1) \prec ... \prec (S^k - S^{k-1}) \text{ and } S^i_{EQ} \prec S^i_{INEQ}$$



Minimum Chain Covering by Dilworth's Theorem



Bipartite Graph Maximum Matching Chain Cover

(1) Construct a bipartite graph from $\mathcal{S} \times \mathcal{S}$

(2) Draw an edge from (S_{EQ}, S_{INEQ}) to (S'_{EQ}, S'_{INEQ}) if $(S_{EQ}, S_{INEQ}) < (S'_{EQ}, S'_{INEQ})$

(3) Compute the maximum cardinality matching on the graph

(4) Map edges from the matching to the minimum chain cover

(5) Create indexes for each chain $\ell = S^1 \prec (S^2 - S^1) \prec ... \prec (S^k - S^{k-1})$ and $S^i_{EQ} \prec S^i_{INEQ}$

Experimental Evaluation in Soufflé

We are looking to compare the performance of R-Tree SPS, B-Tree SPS and the State of the Art

Experimental Setup

- Boost C++ R-Tree implementation for R-Tree SPS
- Soufflé B-Tree implementation for both B-Tree SPS and the State of the Art
- Real-world Datalog applications (DOOP, VPC, DDISASM) with a single thread

Key Metrics

- Compilation Time (s)
- Maximum Memory Usage (KB)
- Evaluation Time (s)

Compilation Time

We find negligible compilation time overhead for both techniques

R-Tree SPS

- DOOP $=5\%$
- VPC $=9\%$
- DDISASM $=5\%$

B-Tree SPS

- DOOP = No Effect
- VPC = No Effect
- DDISASM $=6\%$

R-Tree SPS adds an overhead because of heavy template usage in Boost

B-Tree SPS adds zero compilation time for DOOP and VPC since no extra indexes are built

B-Tree SPS adds 6% to the compilation time for DDISASM since a few extra indexes are built

Maximum Memory Usage

R-tree SPS consumes approximately $2\times$ more memory at peak

B-Tree SPS consumes less than 1% more memory at peak

R-Tree SPS

- DOOP $= 2\%$
- VPC $= 9\%$
- DDISASM $= 137\%$

B-Tree SPS

- DOOP $=$ No Effect
- VPC $=$ No Effect
- DDISASM $= <1\%$

R-Tree SPS consumes more than double the memory at peak since R-Trees must store MBRs

B-Tree SPS consumes zero extra memory for DOOP and VPC since no extra indexes are built

B-Tree SPS consumes less than 1% extra memory for DDISASM since few extra indexes are built

Evaluation Time

R-tree SPS dramatically slows down evaluation time for DOOP and DDISASM

B-Tree SPS dramatically speeds up evaluation time for DDISASM by up to 2.32×

R-Tree SPS

- DOOP $=$ 8.61× to $20.55\times$ *slower*
- VPC $=$ No Effect
- DDISASM $=$ 2.01× to 143.63× *slower*

B-Tree SPS

- DOOP $=$ No Effect
- VPC $=$ No Effect
- DDISASM $=$ 1.02× to 2.32× *faster*

R-Tree SPS dramatically slows down evaluation since searches are $\mathcal{O}(n)$

B-Tree SPS has no effect on DOOP, VPC since there are very few inequalities

B-Tree SPS dramatically speeds up DDISASM since important inequalities are now indexed

Conclusion

Developed a technique that speeds up evaluation of inequalities dramatically

B-Tree SPS vs State of the Art

- Speeds up evaluation time of vital Datalog applications by up to $2.32\times$ (DDISASM)
- Consumes less than 1% more memory at peak
- Increases compilation time by no more than 6%
- Robust for real-world benchmarks with no practical slowdown (DOOP, VPC)
- Technique is entirely automatic and runs in polynomial time
- Demonstrates that index specialisation outperforms index generalisation (R-Tree SPS)

Future Work

Many opportunities for future research

Future Work

- Efficient Evaluation of Min/Max Aggregates
- Improving Performance of Datalog Provenance
- Index Selection with Partial Indexes

References

Molham Aref, Balder ten Cate, Todd J Green, Benny Kimelfeld, Dan Olteanu, Emir Pasalic, Todd L Veldhuizen, and Geoffrey Washburn. (2015).
Design and Implementation of the LogicBlox System, pages 1371-1382.
SIGMOD '15. ACM, New York, NY, USA.

Herbert Jordan, Bernhard Scholz, and Pavle Subotic. (2016).
Soufflé: On Synthesis of Program Analyzers, pages 422-430.
Proceedings of Computer Aided Verification

Pavle Subotic, Herbert Jordan, Lijun Chang, Alan Fekete, and Bernhard Scholz. (2018).
Automatic Index Selection for Large-scale Datalog Computation, pages 141-153.
Proceedings of the VLDB Endowment

Dario Campagna, Beata Sarna-Starosta, and Tom Schrijvers. (2012).
Optimizing Inequality Joins in Datalog with Approximated Constraint Propagation, pages 108-122.
International Symposium on Practical Aspects of Declarative Languages

Antonin Guttman. (1984).
R-trees: A Dynamic Index Structure for Spatial Searching., pages 47-57.
SIGMOD 1984

Pankaj K Agarwal, Jeff Erickson, et al. (1999).
Geometric Range Searching and its Relatives., pages 1-56.
Contemporary Mathematics